INTELLIGIBLE MODELS: RECOVERING LOW DIMENSIONAL ADDITIVE STRUCTURE FOR MACHINE LEARNING MODELS

A Dissertation

Presented to the Faculty of the Graduate School

of Cornell University

in Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy

by

Yin Lou

May 2014

INTELLIGIBLE MODELS: RECOVERING LOW DIMENSIONAL ADDITIVE STRUCTURE FOR MACHINE LEARNING MODELS

Yin Lou, Ph.D.

Cornell University 2014

Different supervised learning models have different bias-variance tradeoffs. For low dimensional problems, low-bias models such boosted trees or SVMs with RBF kernels are very accurate but are unfortunately no longer interpretable by the users. For high dimensional problems, high-bias models such as regularized linear/logistic regressions are usually preferred over other models because of the curse of dimensionality and the exponentially growing hypothesis space but it is not clear whether we could further improve the accuracy from those high-bias models.

Additive modeling is an excellent tool to control the bias and variance in a finer granularity and provides a great solution to these problems. Generalized additive models (GAMs) express the hypothesis as a sum of components, where each component can include any number of variables. Therefore, by prudently selecting the components or restricting the number of complex components and carefully controlling the complexity of each selected component, GAMs are very flexible of modeling hypothesis with different biases.

This dissertation presents a family of additive models called intelligible models, which effectively recover the low dimensional additive structures. Those low dimensional additive components provide the opportunities for data scientists to investigate each simple component individually, and therefore the interpretability is significantly improved. We first present a large-scale empirical study of various methods for fitting GAMs. We demonstrate empirically that gradient boosting with shallow bagged trees yield the best accuracy. In ad-

dition, we propose a very efficient method of detecting pairwise feature interactions that scales to thousands of features. With a large-scale empirical study, we show that models with low dimensional additive components (one- and two-dimensional components) are as accurate as complex models such as random forests. Finally, we develop a method to carefully control the complexity of the intelligible models by feature selection and intelligently deciding whether the selected term is linear or nonlinear, and show that on high dimensional problems we can further improve the accuracy from the popular linear models by allowing a small set of features to act nonlinearly.

BIOGRAPHICAL SKETCH

Yin Lou grew up in Shanghai, China. He had no interaction with computers until his 5th grade in elementary school. Yin started to write programs to solve puzzles during his secondary school soon after he got his first personal computer.

Yin graduated from the high school affiliated to Fudan University in 2005 and obtained his Bachelor of Science degree in Computer Science from Fudan University in 2009. He visited the Hong Kong University of Science and Technology in Fall 2007. Yin started his doctorial program in Computer Science at Cornell University since 2009. Yin's research interests are in the areas of data mining, statistical machine learning, information retrieval, and computer vision. He has been serving as program committee member for International Conference on Machine Learning since 2012.

This document is dedicated to my parents.

ACKNOWLEDGMENTS

This dissertation would not be possible without the help and influence of many people over the years. First of all I would like to thank my advisor, Johannes Gehrke, for his active support and hands-on guidance. Johannes always encouraged me to think about big pictures and aim at solving big problems.

Rich Caruana was my external advisor and I really appreciated the influence that he had on my work. This dissertation started with the discussion with Rich and Johannes while I was interning at Microsoft Research. I was very grateful for his vision on additive modeling and for his superb support and guidance during and after the internship.

I was also very lucky to have had great collaborations with statisticians. I would like to sincerely thank Giles Hooker for providing insightful comments from the point of a statistician and for serving on my thesis committee. I would also like to thank Jacob Bien for guiding me approaching some optimization/theoretical problems and for his generous support and valuable advice. Both of them were actively involved in my research and their expertise in statistics provided additional dimension to this work.

I would also like to thank Noah Snavely for his guidance on computer vision problems. The conversation with Noah has always been inspiring and motivated me for a lot of learning problems with applications on computer vision.

I was very grateful to the members of my thesis committee: Johannes Gehrke, Rich Caruana, Noah Snavely, Giles Hooker and Dexter Kozen. Their constructive feedback and suggestions have helped me improve my work tremendously.

I would also like to thank Nick Craswell at Bing and Daria Sorokina at LinkedIn for their great practical advice on information retrieval and provid-

ing me the opportunity to apply this work on several industry applications.

Finally, I would like to thank my parents for their endless love and support.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 Motivation

Supervised learning, in particular classification and regression, is now widely used in numerous real world applications. Examples of supervised learning based systems include fraudulent transaction detection for credit cards [21], house price prediction [56], learning to rank [43], text categorization [37], detecting objects in images [61], recognizing hand-written digits [12], and so on. A lot of supervised learning models for classification and regression have been developed, such as linear regression [14], logistic regression [51], naïve bayes [53], classification and regression trees [17], random forests [15], gradient boosted regression trees, support vector machines (SVMs) [23] with radial basis function kernel, etc.

Some of those models have demonstrated excellent predictive performance in low dimensional problems, while others are more suitable in high dimensional cases [19, 20], since those models exhibit different bias-variance tradeoff. For example, in many low dimensional problems, complex models such as ensembles of trees, SVMs with RBF kernels are the most accurate since the amount of data is usually enough for effectively modeling the nonlinear effects in the data which cannot be captured by simple models such as linear regression or logistic regression, and therefore these low-bias models often outperform other high-bias models. On the other hand, in high dimensional cases, because of the curse of dimensionality and the exponentially growing hypothesis space, complex low-bias models often overfit and suffer from high variance, while linear models with regularizations tend to offer better predictive performance by

setting up a high bias, which results in a low variance.

However, there are essentially two problems.

- In many applications of low dimension, *what* is learned is just as important as the accuracy of the predictions. Unfortunately, the high accuracy of complex models comes at the expense of interpretability; e.g., even the contribution of individual features to the predictions of a complex model are difficult to understand. *Is there a model which is as accurate as those complex models while still provides ways for data scientists to understand the model by maintaining the intelligibility as much as possible?*
- In high dimensional problems, linear models with regularizations set up a very high bias by restricting each feature to be linear. But we are not sure whether linear model is the best model class to describe or understand the data. *Are there any other alternatives which offer better predictive performance by more carefully balancing bias and variance?*

In this thesis, we revisit the classic additive modeling and show that these two problems can be answered simultaneously by using additive models. we present a family of additive models called *intelligible models*, which effectively recover the low dimensional additive structures. Those low dimensional additive components provide great opportunities for data scientists to investigate each *simple* component individually, and therefore the interpretability is largely preserved. In addition, we present a large-scale empirical study showing that models with low dimensional additive components are as accurate as complex models such as random forests. Finally, we present a method to carefully control the complexity of the intelligible models by feature selection and intelligently deciding whether the selected term is linear or nonlinear, and show that on high dimensional problems we can further improve the accuracy from the

popular linear models by allowing a small set of features to act nonlinearly.

1.2 Contributions and Outline

The organization and contributions of each subsequent chapter are as follows.

Chapter 2 reviews classic additive modeling. We first introduce the standard generalized additive models (GAMs) that employ a one-dimensional smoother to build a restricted class of nonparametric models. Next, we review the classic gradient boosting in the context of additive ensembles. Finally, we discuss the connection of additive models with simple models (such as the standard generalized linear models) and complex models (such as random forests).

Chapter 3 presents our large-scale empirical study of build intelligible models using GAMs. In this chapter, we consider various combinations of shape functions (a.k.a one-dimensional smoothers or base learners) and optimization algorithms [46]. Since the shape functions can be arbitrarily complex, GAMs are more accurate than simple linear models. But since they do not contain any interactions between features, they can be easily interpreted by users. Our study includes existing spline and tree-based methods for shape functions and penalized least squares, gradient boosting, and backfitting for learning GAMs. We also present a new method based on tree ensembles with an adaptive number of leaves that consistently outperforms previous work. We complement our experimental results with a bias-variance analysis that explains how different shape models influence the additive model.

Chapter 4 extends our previous study and proposes GA^{2}M-models, for *Generalized Additive Models plus Interactions* [47]. In this chapter, we suggest adding selected terms of interacting *pairs* of features to standard GAMs. Since these models only include one- and two-dimensional components, the components of

GA^{2}M-models can be visualized and interpreted by users. To explore the huge (quadratic) number of pairs of features, we develop a novel, computationally efficient method called FAST for ranking all possible pairs of features as candidates for inclusion into the model. In a large-scale empirical study, we show the effectiveness of FAST in ranking candidate pairs of features. In addition, we show the surprising result that GA^{2}M have almost the same predictive performance as the best full-complexity models on a number of real datasets. This postulates that for many problems, using one- and two-dimensional additive components are enough to yield models that are both intelligible and accurate.

Chapter 5 studies the problem of complexity control of GAMs. In this chapter, we introduce generalized sparse partially linear additive models (SPLAMs), which in addition to variable selection, each variable can stay linear as in standard generalized linear models (GLMs), or it can be allowed to act nonlinearly as in standard generalized additive models (GAMs) when there is sufficient evidence in the data. We model the problem as a convex hierarchical sparse regularization problem and propose two algorithms using block coordinate gradient descent and block coordinate descent, respectively, to solve the optimization problem. A statistical analysis of the theoretical properties for SPLAM is provided. A general technique for optimization on data-sparse problem is also discussed. In this study, we aim to provide alternatives to ℓ_1 regularized linear models in high dimensional cases. Our large-scale experiments show that SPLAM effectively recovers the underlying additive structure and often outperforms GLM/GAM in terms of accuracy.

Chapter 6 concludes this thesis and presents suggestions for future work.

CHAPTER 2

ADDITIVE MODELING

In this chapter, we review the classic additive modeling. We first introduce the standard generalized additive models (GAMs) that employ a one-dimensional smoother to build a restricted class of nonparametric models. Next, we review the classic boosting algorithms in the context of additive ensembles. Finally, we discuss the connection of additive models with simple models (such as the standard generalized linear models) and complex models (such as random forests).

Let $\mathcal{D} = \{(\boldsymbol{x}_i, y_i)\}_1^N$ denote a training dataset of size N, where $\boldsymbol{x}_i = (x_{i1}, ..., x_{ip})$ is a feature vector with p features and y_i is the target (response). We use x_j to denote the jth variable in the feature space. Let L denote some loss function, such as squared loss, or logistic loss.

2.1 Generalized Additive Models

Generalized additive models (GAMs) are nonparametric methods which use a one-dimensional smoother to build a restricted class of nonparametric models. GAMs take the following standard form.

$$g(E[y]) = \sum_j f_j(x_j) \tag{2.1}$$

The function $g(\cdot)$ is called the *link function* and we call the f_is *shape functions* (a.k.a smoothers or base learners). If the link function is the identity function, Equation 2.3 describes an additive model (e.g., a regression model); if the link function is the logit function, Equation 2.3 describes a generalized additive model (e.g., a classification model). These two link functions are the most common ones used in most real applications.

Algorithm 1 Backfitting for Regression

1: $f_j \leftarrow 0$
2: **for** $m = 1$ to M **do**
3: **for** $j = 1$ to p **do**
4: $\mathcal{R} \leftarrow \{x_{ij}, y_i - \sum_{k \neq j} f_k\}_1^N$
5: Learn shaping function $S : x_j \rightarrow y$ using $\mathcal{R}$ as training dataset
6: $f_j \leftarrow S$

Classic approach to learning additive models is the backfitting algorithm [32]. The algorithm starts with an initial guess of all shape functions (such as setting them all to zero). The first function f_1 is then learned using the training set with the goal to predict y. Then we learn the second shape function f_2 on the residuals $y - f_1(x_1)$, i.e., using training set $\{(x_{i2}, y_i - f_1(x_{i1}))\}_1^N$. The third shape function is trained on the residuals $y - f_1(x_1) - f_2(x_2)$, and so on. After we have trained all p shape functions, the first shape function is *discarded* and retrained on the residuals of the other $p - 1$ shape functions. Note that backfitting is a functional form of the "Gauss-Seidel" algorithm and its convergence is usually guaranteed [32]. Algorithm 1 summarizes the backfitting algorithm.

For classification problems, the classic approach is to use a generalized version of the backfitting algorithm called the "Local Scoring Algorithm" [32], as summarized in Algorithm 2 . Let $F(\boldsymbol{x}) = \sum_j f_j(x_j)$, we form the working response (Line 3)

$$\tilde{y}_i = F(\boldsymbol{x}_i) + \frac{\mathbf{1}(y_i = 1) - p(\boldsymbol{x}_i)}{p(\boldsymbol{x}_i)(1 - p(\boldsymbol{x}_i))},$$

where $p(\boldsymbol{x}_i) = \frac{1}{1+\exp(-F(\boldsymbol{x}_i))}$. We then apply the weighted backfitting algorithm to the response $\tilde{y}_i$ with observation weights $p(\boldsymbol{x}_i)(1 - p(\boldsymbol{x}_i))$ [32] (Line 4-5).

Algorithm 2 Backfitting for Classification

1: $f_j \leftarrow 0$
2: **for** $m = 1$ to M **do**
3: $\quad \tilde{y}_i = F(\boldsymbol{x}_i) + \frac{\mathbf{1}(y_i=1)-p(\boldsymbol{x}_i)}{p(\boldsymbol{x}_i)(1-p(\boldsymbol{x}_i))}$, for $i = 1, ..., N$
4: $\quad w_i \leftarrow p(\boldsymbol{x}_i)(1 - p(\boldsymbol{x}_i))$
5: $\quad$ Apply weighted backfitting algorithm in Algorithm 1

2.2 Additive Ensembles

In this section, we describe gradient boosted tree ensembles [25, 26]. Gradient boosted trees are additive ensembles which greedily build small steps (trees) to fit the data. The final prediction is the sum of all functions in the ensemble. Since gradient boosting often uses regression trees as base learners, it is also called gradient boosted regression trees (GBRT). GBRT has been widely used in a lot of industry-scale real applications.

Gradient boosting framework [26] is described in Algorithm 3. Given the loss function L, gradient boosting adopts a "greedy stagewise" approach to build an additive function F_M,

$$F_M(\boldsymbol{x}) = \sum_{m=1}^{M} \rho_m h(\boldsymbol{x}; \boldsymbol{a}_m),$$

such that, at each stage m,

$$\{\rho_m, \boldsymbol{a}_m\} = \arg\min_{\rho, \boldsymbol{a}} \sum_{i=1}^{N} L\left(y_i, F_{m-1} + \rho h(\boldsymbol{x}_i; \boldsymbol{a})\right)$$

Here $h(\boldsymbol{x}; \boldsymbol{a})$ is a "weak" learner. Instead of directly solving this difficult problem, [26] approximately conducted steepest descent in function space by solving a least square problem (Line 4),

$$\boldsymbol{a}_m = \arg\min_{\boldsymbol{a}, \rho} \sum_{i=1}^{N} [-g_m(\boldsymbol{x}_i) - \rho h(\boldsymbol{x}_i; \boldsymbol{a})]^2,$$

where

$$-g_m(\boldsymbol{x}_i) = -\left[\frac{\partial L(y_i, F(\boldsymbol{x}_i))}{\partial F(\boldsymbol{x}_i)}\right]_{F(\boldsymbol{x})=F_{m-1}(\boldsymbol{x})}$$

Algorithm 3 Gradient Boosting

1: $F_0(\boldsymbol{x}) \leftarrow \arg\min_\rho \sum_{i=1}^N L(y_i, \rho)$
2: **for** $m = 1$ to M **do**
3: $\quad \tilde{y}_i \leftarrow -\left[\frac{\partial L(y_i, F(\boldsymbol{x}_i))}{\partial F(\boldsymbol{x}_i)}\right]_{F(\boldsymbol{x})=F_{m-1}(\boldsymbol{x})}, i = 1, ..., N$
4: $\quad \boldsymbol{a}_m \leftarrow \arg\min_{\boldsymbol{a},\rho} \sum_{i=1}^N [\tilde{y}_i - \rho h(\boldsymbol{x}_i; \boldsymbol{a})]^2$
5: $\quad \rho_m \leftarrow \arg\min_\rho \sum_{i=1}^N L\left(y_i, F_{m-1}(\boldsymbol{x}_i) + \rho h(\boldsymbol{x}_i; \boldsymbol{a}_m)\right)$
6: $\quad F_m(\boldsymbol{x}) \leftarrow F_{m-1}(\boldsymbol{x}) + \rho_m h(\boldsymbol{x}; \boldsymbol{a}_m)$

Algorithm 4 GBRT for Regression

1: $F_0(\boldsymbol{x}) \leftarrow \bar{y}$
2: **for** $m = 1$ to M **do**
3: $\quad \tilde{y}_i \leftarrow y_i - F_{m-1}(\boldsymbol{x}_i), i = 1, ..., N$
4: $\quad \{R_{jm}\}_1^J \leftarrow$ a J-terminal node tree trained on $\{(\boldsymbol{x}_i, \tilde{y}_i)\}_1^N$
5: $\quad F_m(\boldsymbol{x}) \leftarrow F_{m-1}(\boldsymbol{x}) + \sum_{j=1}^J \gamma_{jm} \mathbf{1}(\boldsymbol{x} \in R_{jm})$

is the steepest descent direction in the N-dimensional data space at $F_{m-1}(\boldsymbol{x})$. For the other coefficient ρ_m, a line search is performed (Line 5),

$$\rho_m \leftarrow \arg\min_\rho \sum_{i=1}^N L\left(y_i, F_{m-1}(\boldsymbol{x}_i) + \rho h(\boldsymbol{x}_i; \boldsymbol{a}_m)\right).$$

Algorithm 4 describes the GBRT algorithm for regression problems. At each iteration, gradient boosting forms the current residual (Line 3) and builds a J-terminal node regression tree with γ_j denoting the prediction value for leaf R_j(Line 4), which is then added to the ensemble (Line 5).

Algorithm 5 describes the GBRT algorithm for binary classification problems. Here the loss function is the negative binomial log-likelihood,

$$L(y, F) = \log(1 + \exp(-2yF)), y \in \{-1, 1\}$$

The pseudo response is (Line 3),

$$\tilde{y}_i = -\left[\frac{\partial L(y_i, F(\boldsymbol{x}_i))}{\partial F(\boldsymbol{x}_i)}\right]_{F(\boldsymbol{x})=F_{m-1}(\boldsymbol{x})} = \frac{2y_i}{1 + \exp(2y_i F(\boldsymbol{x}_i))}$$

The line search becomes

$$\rho_m = \arg\min \sum_{i=1}^N \log(1 + \exp(-2y_i(F_{m-1}(\boldsymbol{x}_i) + \rho h(\boldsymbol{x}_i; \boldsymbol{a}_m))))$$

Algorithm 5 GBRT for Classification

1: $F(\boldsymbol{x}_i) \leftarrow 0$
2: **for** $m = 1$ to M **do**
3: $\quad \tilde{y}_i \leftarrow \frac{2y_i}{1+\exp(2y_i F(\boldsymbol{x}_i))}, i = 1, ..., N$
4: $\quad \{R_{jm}\}_1^J \leftarrow$ a J-terminal node tree trained on $\{(\boldsymbol{x}_i, \tilde{y}_i)\}_1^N$
5: $\quad \gamma_{jm} = \frac{\sum_{x_{ij} \in R_{jm}} \tilde{y}_i}{\sum_{x_{ij} \in R_{jm}} |\tilde{y}_i|(2-|\tilde{y}_i|)}, j = 1, ..., J$
6: $\quad F(\boldsymbol{x}_i) \leftarrow F(\boldsymbol{x}_i) + \sum_{j=1}^J \gamma_{jm} \mathbf{1}(\boldsymbol{x}_i \in R_{jm})$

With J-terminal node regression tree as base learners (Line 4), we use the strategy of separate updates in each terminal node R_{jm},

$$\gamma_{jm} = \arg\min_{\gamma} \sum_{\boldsymbol{x}_i \in R_{jm}} \log(1 + \exp(-2y_i(F_{m-1}(\boldsymbol{x}_i) + \gamma))). \tag{2.2}$$

There is no closed form solution to 2.2, we approximate it by a single Newton-Raphon step (Line 5),

$$\gamma_{jm} = \sum_{\boldsymbol{x}_i \in R_{jm}} \tilde{y}_i / \sum_{\boldsymbol{x}_i \in R_{jm}} |\tilde{y}_i|(2 - |\tilde{y}_i|)$$

Finally the updated tree is added to the ensemble (Line 6).

The gradient boosting algorithm is deterministic, stochastic gradient boosting further improves the predictive performance by adding randomness at each boosting iteration [27]. In particular, at each boosting iteration, a bootstrap sample is drawn and the standard gradient boosting step is applied on the sampled data instead of the full data.

Another popular boosted tree algorithm for classification problems is LogitBoost [25]. LogitBoost differs from GBRT in that LogitBoost forms the pseudo residual by taking the point-wise Newton gradient. As shown in Algorithm 6, the Newton-Raphson step is pushed before the construction of the tree (Line 3), where $y_i^* = 1$ if $y_i = 1$ and $y_i^* = 0$ when $y_i = -1$. A weight is assigned to each point, and a J-leaf regression tree is built on this weighted training set with the pseudo responses (Line 4). As a result, no post-Newton step is needed to update

Algorithm 6 LogitBoost for Classification

1: Start with weights $w_i = 1/N$, $i = 1, ..., N$, $F(\boldsymbol{x}) = 0$ and probability estimates $p(\boldsymbol{x}_i) = 1/2$.
2: **for** $m = 1$ to M **do**
3: $\quad z_i \leftarrow \frac{y_i^* - p(\boldsymbol{x}_i)}{p(\boldsymbol{x}_i)(1-p(\boldsymbol{x}_i))}, i = 1, ..., N$
4: $\quad w_i \leftarrow p(\boldsymbol{x}_i)(1 - p(\boldsymbol{x}_i))$
5: $\quad f_m(\boldsymbol{x}) \leftarrow$ a J-leaf tree trained on $\{(\boldsymbol{x}_i, z_i)\}_1^N$ using weights $\{w_i\}_1^N$
6: $\quad F(\boldsymbol{x}_i) \leftarrow F(\boldsymbol{x}_i) + \frac{1}{2} f_m(\boldsymbol{x})$
7: $\quad p(\boldsymbol{x}) \leftarrow \frac{\exp(F(\boldsymbol{x}))}{\exp(F(\boldsymbol{x})) + \exp(-F(\boldsymbol{x}))}$

the leaf values. Li *et al.* showed that LogitBoost exhibits better empirical convergence than GBRT [42], while Friedman pointed out the weighting scheme in LogitBoost can sometimes cause numerical instability [26].

2.3 Discussion

A special class of GAMs is called generalized linear molds (GLMs), which have the following form.

$$g(E[y]) = \beta_0 + \sum_j \beta_j x_j \tag{2.3}$$

It is clear that standard GAMs subsume the class of GLMs by forcing each component to be linear. Therefore, GAMs have lower bias than GLMs.

On the other hand, when each component of GAMs can use all features, the model class is equivalent to the following.

$$y = f(x_1, ..., x_p) \tag{2.4}$$

This is a special form of additive models in which only one component is in the model. Complex methods such as classification and regression trees, random forests, etc., directly build models of this form.

In addition, we can consider a full additive model with all possible compo-

nents as follows.

$$g(E[y]) = \sum_j f_j(x_j) + \sum_{ij} f_{ij}(x_i, x_j) + \sum_{ijk} f(x_i, x_j, x_k) + ... \tag{2.5}$$

Components involving more than one features are called statistical interactions. By adding higher order interactions, we gradually move from standard GAMs to full complexity models such as random forests. As we will see later in Chapter 4, adding pairwise interactions is usually enough to build accurate models while maintaining the intelligibility as much as possible.

CHAPTER 3

INTELLIGIBLE MODELS FOR CLASSIFICATION AND REGRESSION

Everything should be made as simple as possible, but not simpler.

— Albert Einstein.

3.1 Introduction

Classification and regression are two of the most important data mining/machine learning tasks. Currently, the most accurate methods on many datasets are complex models such as boosted trees, SVMs, or deep neural nets. However, in many applications *what* is learned is just as important as the accuracy of the predictions. Unfortunately, the high accuracy of complex models comes at the expense of interpretability; e.g., even the contribution of individual features to the predictions of a complex model are difficult to understand.

The goal of this work is to construct accurate models that are interpretable. By interpretability we mean that users can understand the contribution of individual features in the model; e.g., we want models that can quantify the impact of each predictor. This desiderata permits arbitrary complex relationships between individual features and the target, but excludes models with complex interactions between features. Thus in this chapter we fit models of the form:

$$g(E[y]) = f_1(x_1) + ... + f_n(x_n), \tag{3.1}$$

which are known as *generalized additive models* in the literature [32, 68]. The function $g(\cdot)$ is called the *link function* and we call the f_is *shape functions*. If the link function is the identity, Equation 3.1 describes an additive model (e.g., a regression model); if the link function is the logit function, Equation 3.1 describes a generalized additive model (e.g., a classification model).

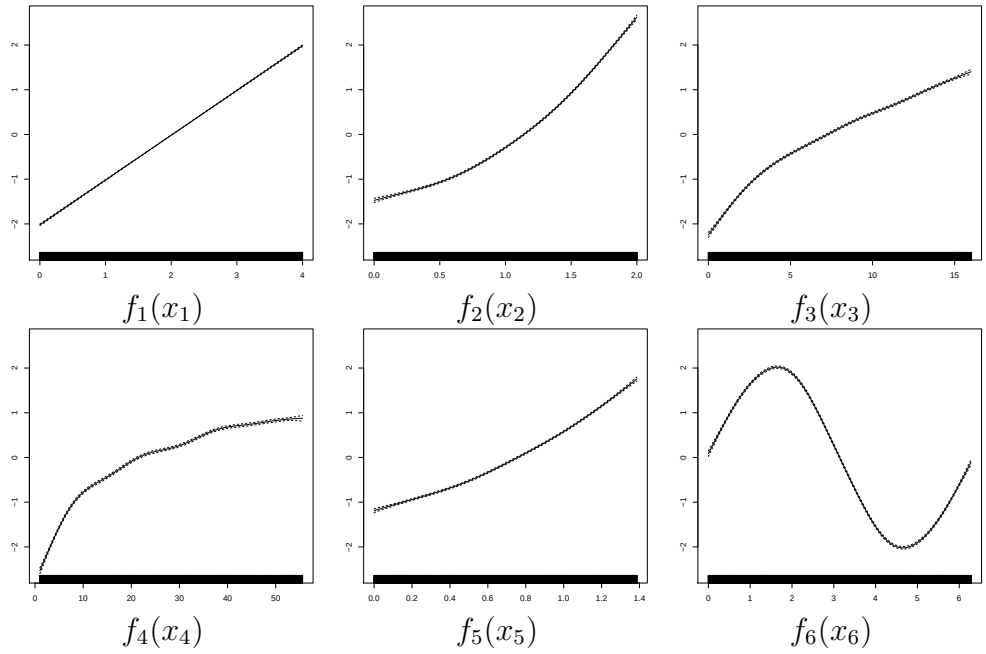


Figure 3.1: Shape Functions for Synthetic Dataset in Example 1.

Example 1. *Assume we are given a dataset with 10,000 points generated from the model* $y = x_1 + x_2^2 + \sqrt{x_3} + \log(x_4) + \exp(x_5) + 2\sin(x_6) + \epsilon$, *where* $\epsilon \sim \mathcal{N}(0, 1)$. *After fitting an additive model to the data of the form shown in Equation 3.1, we can visualize the contribution of* x_i*s as shown in Figure 3.1: Because predictions are a linear function of the* $f_i(x_i)$*, scatterplots of* $f_i(x_i)$ *on the y-axis vs.* x_i *on the x-axis allow us to visualize the shape function that relates the* $f_i(x_i)$ *to the* x_i*, thus we can easily understand the contribution of* x_i *to the prediction.*

Because the data in Example 1 was drawn from a model with no interactions between features, a model of the form in Equation 3.1 is able to fit the data perfectly (modulo noise). However, data are not always so simple in practice. As a second example, consider a real dataset where there may be interactions between features.

Example 2. *The "Concrete" dataset relates the compressive strength of concrete to its age and ingredients. The dataset contains 1030 points with eight numerical features.*

Table 3.1: From Linear to Additive Models.

Model	Form	Intelligibility	Accuracy
Linear Model	$y = \beta_0 + \beta_1 x_1 + ... + \beta_n x_n$	+++	+
Generalized Linear Model	$g(y) = \beta_0 + \beta_1 x_1 + ... + \beta_n x_n$	+++	+
Additive Model	$y = f_1(x_1) + ... + f_n(x_n)$	++	++
Generalized Additive Model	$g(y) = f_1(x_1) + ... + f_n(x_n)$	++	++
Full Complexity Model	$y = f(x_1, ..., x_n)$	+	+++

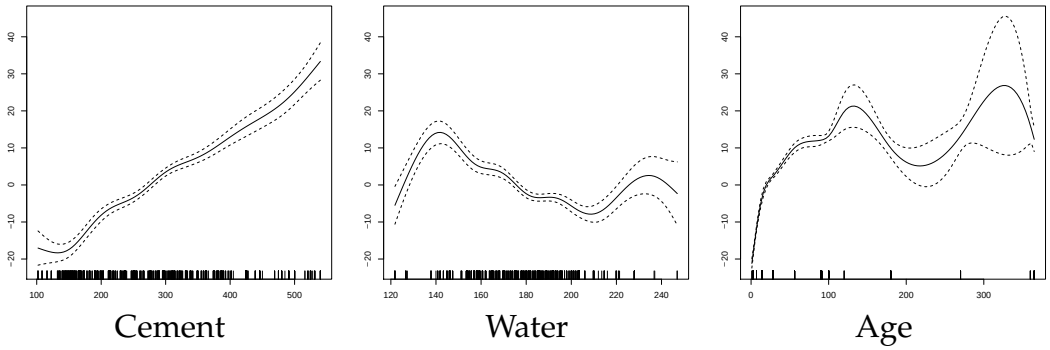


Figure 3.2: Shape Functions for Concrete Dataset in Example 2.

We again fit an additive model of the form in Equation 3.1. Figure 3.2 shows scatter plots of the shape functions learned for three of the eight features. As we can see from the figure, the compressibility of concrete depends nearly linearly on the Cement feature, but it is a complex non-linear function of the Water and Age features; we say that the model has shaped *these features. A linear model without the ability to shape features would have worse fit because it cannot capture these non-linearities. Moreover, an attempt to interpret the contribution of features by examining the slopes of a simple linear model would be misleading; the additive model yields much better fit to the data while still remaining intelligible.*[1]

As we saw in the examples, additive models explicitly decompose a complex function into one-dimensional components, its shape functions. Note, however, that the shape functions themselves may be non-linear: Each feature x_i can have

[1]See Section 3.4 for the fit of different models to this dataset.

a complex non-linear shape $f_i(x_i)$, and thus the accuracy of additive models can be significantly higher than the accuracy of simple linear models. Table 3.1 summarizes the differences between models of different complexity that we consider in this chapter. Linear models, and generalized linear models (GLMs) are the most intelligible, but often the least accurate. Additive models, and generalized additive models (GAMs) are more accurate than GLMs on many data sets because they capture non-linear relationships between (individual) features and the response, but retain much of the intelligibility of linear models. Full complexity models such as ensembles of trees are more accurate on many datasets because they model both non-linearity and interaction, but are so complex that it is nearly impossible to interpret them.

In this chapter we present the results of (to the best of our knowledge) the largest experimental study of GAMs. We consider shape functions based on splines [68] and boosted stumps [26], as well as novel shape functions based on bagged and boosted ensembles of trees that choose the number of leaves adaptively. We experiment with (iteratively re-weighted) least squares, gradient boosting, and backfitting to both iteratively refine the shape functions and construct the linear model of the shaped features. We apply these methods to six classification and six regression tasks. For comparison, we fit simple linear models as a baseline. We also fit unrestricted ensembles of trees as full complexity models to get an idea of what accuracy is achievable.

Table 3.2 summarizes the key findings of our study. Entries in the table are the average accuracies on the regression and classification datasets, normalized by the accuracy of Penalized (Iteratively Re-weighted) Least Squares with Splines (P-LS/P-IRLS). As expected, the accuracy of GAMs falls between that of linear/logistic regression without feature shaping and full-complexity models

Table 3.2: Preview of Empirical Results.

Model	Regression	Classification	Mean
Linear/Logistic	1.68	1.22	1.45
P-LS/P-IRLS	1.00	1.00	1.00
BST-SP	1.03	1.00	1.02
BF-SP	1.00	1.00	1.00
BST-bagTR2	0.96	0.96	0.96
BST-bagTR3	0.97	0.94	0.96
BST-bagTR4	0.99	0.95	0.97
BST-bagTRX	0.95	0.94	0.95
Random Forest	0.88	0.80	0.84

such as random forests. However, surprisingly, the best GAM models have accuracy much closer to the full-complexity models than to the linear models. Our results show that bagged trees with 2-4 leaves as shape functions in combination with gradient boosting as learning method (Methods BST-bag-TR2 to BST-bag-TR4) outperform all other methods on most datasets. Our novel method of adaptively selecting the right number of leaves (Method BST-bagTRX) is almost always even better, and thus we recommend it as the method of choice. On average, this method reduces loss by about 5% over previous GAM models, a significant improvement in practice.

The rest of the chapter is structured as follows. Section 3.2 presents algorithms for fitting generalized additive models with various shape functions and learning methods. Section 3.3 describes our experimental setup, Section 3.4 presents the results and their interpretation, followed by a discussion in Section 3.5 and an overview of related work in Section 3.6. We conclude in Section 3.7.

3.2 Methodology

Let $\mathcal{D} = \{(\boldsymbol{x}_i, y_i)\}_1^N$ denote a training dataset of size N, where $\boldsymbol{x}_i = (x_{i1}, ..., x_{in})$ is a feature vector with n features and y_i is the target. In this chapter, we con-

sider both regression problems where $y_i \in \mathbb{R}$ and binary classification problems where $y_i \in \{1, -1\}$. Given a model F, let $F(\boldsymbol{x}_i)$ denote the prediction of the model for data point $\boldsymbol{x}_i$. Our goal in both classification and regression is to minimize the expected value of some loss function $L(y, F(\boldsymbol{x}))$.

We are working with generalized additive models of the form in Equation 3.1. To train such models we have to select (i) the shape functions for individual features and (ii) the learning method used to train the overall model. We discuss these two choices next.

3.2.1 Shape Functions

In our study we consider two classes of shape functions: regression splines and trees or ensembles of trees. Note that all shape functions relate a single attribute to the target.

Regression Splines. We consider regression splines of degree d of the form $y = \sum_{k=1}^{d} \beta_k b_k(x)$.

Trees and Ensembles of Trees. We also use binary trees and ensembles of binary trees with largest variance reduction as split selection method. We control tree complexity by either fixing the number of leaves or by disallowing leaves that have fewer than an α-fraction of the number of training examples.

We consider the following ensemble variants:

- Single Tree. We use a single regression tree as a shape function.
- Bagged Trees. We use the well-known technique of bagging to reducing variance [10].
- Boosted Trees. We use gradient boosting, where each successive tree tries to predict the overall residual from all preceding trees [26].

- Boosted Bagged Trees. We use a bagged ensemble in each step of gradient boosting [26], resulting in a boosted ensemble of bagged trees.

3.2.2 Generalized Additive Models

We consider three different methods for fitting additive models in our study: Least squares fitting for learning regression spline shape functions, and gradient boosting and backfitting for learning tree and tree ensemble shape functions. We review them here briefly for completeness although we would like to emphasize that these methods are not a contribution of this thesis.

Least Squares

Fitting a spline reduces to learning the weights $\beta_k(x)$ for the basis functions $b_k(x)$. Learning the weights can be reduced to fitting a linear model $y = \boldsymbol{X}\boldsymbol{\beta}$, where $\boldsymbol{X}_i = [b_1(x_{i1}), ..., b_k(x_{in})]$; the coefficients of the linear model can be computed exactly using the least squares method [68]. To control smoothness, there is a "wiggliness" penalty: we minimize $\|y - \boldsymbol{X}\boldsymbol{\beta}\| + \lambda \sum_i \int [f_i''(x_i)]^2 dx$ with the smoothing parameter λ. Large values of λ lead to a straight line for f_i while low values of λ allow the spline to fit closely to the data. We use thin plate regression splines from the R package "mgcv" [68] that automatically selects the best values for the parameters of the splines [67]. We call this method penalized least-squares (P-LS) in our experiments.

The fitting of an additive logistic regression model using splines is similarly reduced to fitting a logistic regression with a different basis, which can be solved using penalized-iteratively reweighted least squares (P-IRLS) [68].

Gradient Boosting

We use standard gradient boosting [26, 27] with one difference: Since we want to learn shape functions for all features, in each iteration of boosting we cycle sequentially through all features. For completeness, we include pseudo-code in Algorithms 7 and 8. In Algorithm 7, we first set all shape functions to zero (Line 1). Then we loop over M iterations (Line 2) and over all features (Line 3) and then calculate the residuals (Line 4). We then learn then one-dimensional function to predict the residuals (Line 5) and add it to the shape function (Line 6).

Backfitting

A popular algorithm for learning additive models is the backfitting algorithm [32]. The algorithm starts with an initial guess of all shape functions (such as setting them all to zero). The first shape function f_1 is then learned using the training set with the goal to predict y. Then we learn the second shape function f_2 on the residuals $y - f_1(x_1)$, i.e., using training set $\{(x_{i2}, y_i - f_1(x_{i1}))\}_1^N$. The third shape function is trained on the residuals $y - f_1(x_1) - f_2(x_2)$, and so on. After we have trained n shape functions, the first shape function is *discarded* and retrained on the residuals of the other $n - 1$ shape functions. Note that backfitting is a form of the "Gauss-Seidel" algorithm and its convergence is usually guaranteed [32]. Its pseudocode looks identical to Algorithm 7 except that pseudo residual is formed by excluding current f_j in Line 4 and Line 6 is replaced by $f_j \leftarrow S$.

To fit an additive logistic regression model, we can use a generalized version of the backfitting algorithm called the "Local Scoring Algorithm" [32], which is

Algorithm 7 Gradient Boosting for Regression

$f_j \leftarrow 0$
for $m = 1$ to M **do**
  for $j = 1$ to n **do**
    $\mathcal{R} \leftarrow \{x_{ij}, y_i - \sum_k f_k\}_1^N$
    Learn shaping function $S : x_j \rightarrow y$ using $\mathcal{R}$ as training dataset
    $f_j \leftarrow f_j + S$

Algorithm 8 Gradient Boosting for Classification

$f_j \leftarrow 0$
for $m = 1$ to M **do**
  for $j = 1$ to n **do**
    $\tilde{y}_i \leftarrow \frac{2y_i}{1+\exp(2y_i F(\boldsymbol{x}_i))}, i = 1, ..., N$
    Learn $\{R_{km}\}_1^K \leftarrow$ a tree with K leaf nodes using $\{(x_{ij}, \tilde{y}_i)\}_1^N$ as training dataset
    $\gamma_{km} = \frac{\sum_{x_{ij} \in R_{km}} \tilde{y}_i}{\sum_{x_{ij} \in R_{km}} |\tilde{y}_i|(2-|\tilde{y}_i|)}, k = 1, ..., K$
    $f_j \leftarrow f_j + \sum_{k=1}^K \gamma_{km} \mathbf{1}(x_{ij} \in R_{km})$

a general method for fitting generalized additive models. We form the response

$$\tilde{y}_i = F(\boldsymbol{x}_i) + \frac{\mathbf{1}(y_i = 1) - p(\boldsymbol{x}_i)}{p(\boldsymbol{x}_i)(1 - p(\boldsymbol{x}_i))},$$

where $p(\boldsymbol{x}_i) = \frac{1}{1+\exp(-F(\boldsymbol{x}_i))}$. We then apply the weighted backfitting algorithm to the response $\tilde{y}_i$ with observation weights $p(\boldsymbol{x}_i)(1 - p(\boldsymbol{x}_i))$ [32].

3.3 Experimental Setup

In this section we describe the experimental design.

3.3.1 Datasets

We selected datasets of low-to-medium dimensionality with at least 1000 points. Table 3.3 summarizes the characteristics of the 12 datasets. One of the regression datasets is a synthetic problem used to illustrate feature shaping (but we do not use the results on this dataset when comparing the accuracy of the methods).

Table 3.3: Datasets.

Dataset	Size	Attributes	%Pos
Concrete	1030	9	-
Wine	4898	12	-
Delta	7192	6	-
CompAct	8192	22	-
Music	50000	90	-
Synthetic	10000	6	-
Spambase	4601	58	39.40
Insurance	9823	86	5.97
Magic	19020	11	64.84
Letter	20000	17	49.70
Adult	46033	9/43	16.62
Physics	50000	79	49.72

The "Concrete," "Wine," and "Music" regression datasets are from the UCI repository [1]; "Delta" is the task of controlling the ailerons of a F16 aircraft [2]; "CompAct" is a regression dataset from the Delve repository that describes the state of multiuser computers [3]. The synthetic dataset was described in Example 1.

The "Spambase," "Insurance," "Magic," "Letter" and "Adult" classification datasets are from the UCI repository. "Adult" contains nominal attributes that we transformed to boolean attributes (one boolean per value). "Letter" has been converted to a binary problem by using A-M as positives and the rest as negatives. The "Physics" dataset is from the KDD Cup 2004 [4].

3.3.2 Methods

Recall from Section 3.2 that we have two different types of shape functions and three different methods of learning generalized additive models; see Table 3.4 for an overview of these methods and their names. While penalized least squares for regression (P-LS) and penalized iteratively re-weighted least squares

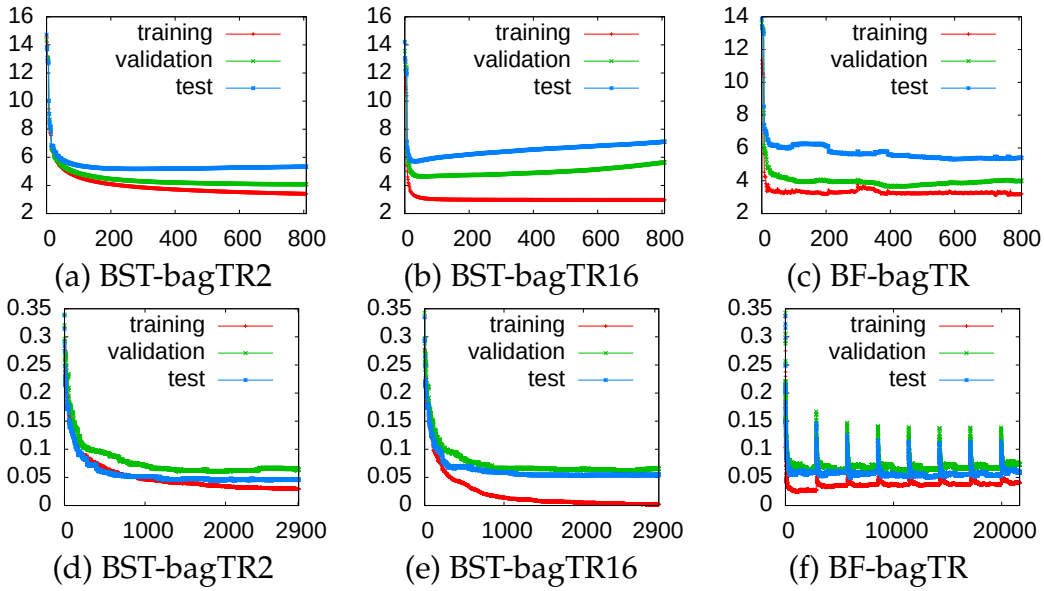


Figure 3.3: Training curves for gradient boosting and backfitting. Figure (a), (b) and (c) show the behavior of BST-bagTR2, BST-bagTR16 and BF-bagTR on the "Concrete" regression problem, respectively. Figure (d), (e) and (f) illustrate behavior of BST-bagTR2, BST-bagTR16 and BF-bagTR on the "Spambase" classification, respectively.

for classification (P-IRLS) only work with splines, gradient boosting and backfitting can be applied to both splines and ensembles of trees.

In gradient boosting, we vary the number of leaves in the bagged or boosted trees: $2, 3, 4, 8, 12$ to 16 (indicated by appending this number to the method names). Trained models will contain M such trees for each shape function after M iterations. In backfitting, we re-build the shape function for each feature from scratch in each round, so the shape function needs to have enough expressive power to capture a complex function. Thus we control the complexity of the tree not by the number of leaves, but by adaptively choosing a parameter α that stops splitting nodes smaller than an α fraction of the size of the training data; we vary $\alpha \in \{0.00125, 0.025, 0.05, 0.1, 0.15, 0.2, 0.25, 0.3, 0.35, 0.4, 0.45, 0.5\}$. A summary of the combinations of shape functions and learning methods can be found in Table 3.4.

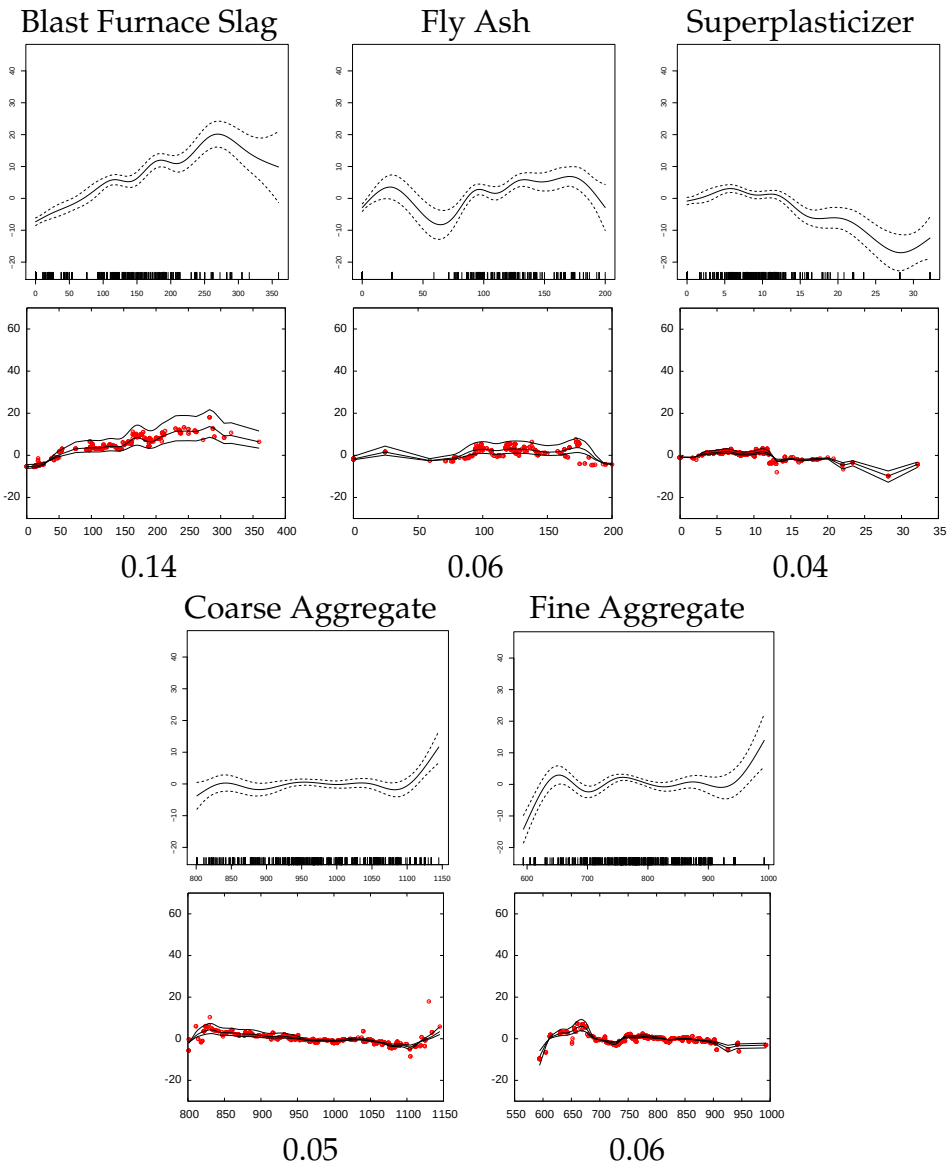


Figure 3.4: Shapes of features for the "Concrete" dataset produced by P-LS (top) and BST-bagTR3 (bottom).

Beyond the parameters that we already discussed, P-LS and P-IRLS have a parameter λ, which is estimated using generalized cross validation as discussed in Section 3.2. We do not fix the number of iterations for gradient boosting and backfitting but instead run these methods until convergence as follows: We divide the training set into five partitions. We then set aside one of the partitions

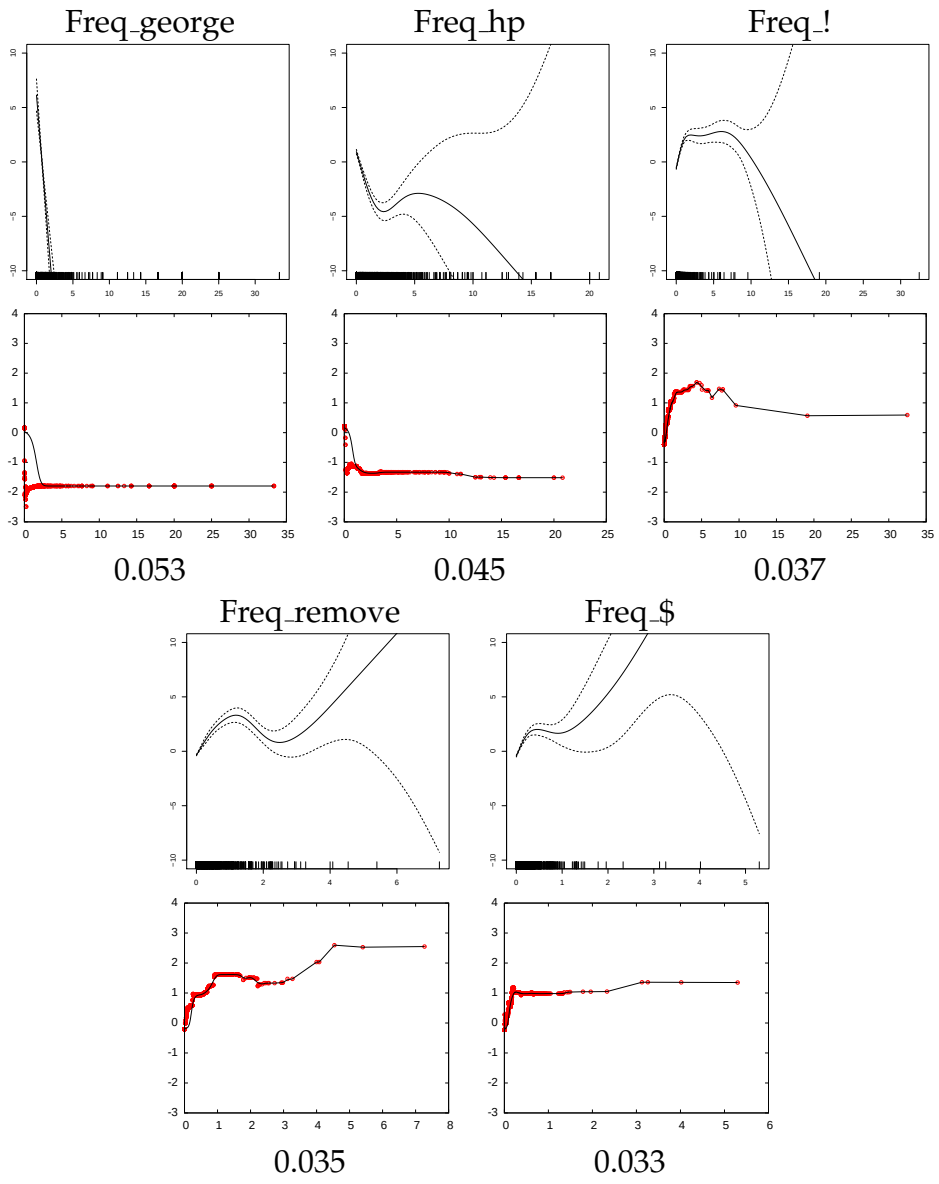


Figure 3.5: Shapes of features for the "Spambase" dataset produced by P-IRLS (top) and BST-bagTR3 (bottom).

as a validation set, train the model on the remaining four partitions, and use the validation set to check for convergence. We repeat this process five times and then compute M, the average number of iterations until convergence across the five iterations. We then re-train the model using the whole training set for M iterations. We follow a similar procedure for backfitting where we pick the best

Table 3.4: Notation for learning methods and shape functions.

Shape Function	Least Squares	Gradient Boosting	Backfitting
Splines	P-LS/P-IRLS	BST-SP	BF-SP
Single Tree	N/A	BST-TRx	BF-TR
Bagged Trees	N/A	BST-bagTRx	BF-bagTR
Boosted Trees	N/A	BST-TRx	BF-bstTRx
Boosted Bagged Trees	N/A	BST-bagTRx	BF-bbTRx

α for each partition and average them to train the final model using the whole training dataset.

3.3.3 Metrics

For regression problems, we report the root mean squared error (RMSE) for linear regression (no feature shaping), additive models with shaping with splines or trees (penalized least squares, gradient boosting and backfitting), and unrestricted full-complexity models (random forest regression trees and Additive Groves [5, 59]).

For classification problems, we report the error rates for logistic regression, generalized additive models with splines or trees (penalized iteratively reweighted least squares, gradient boosting and backfitting), and full-complexity unrestricted models (random forests [15]).[2]

In all experiments we use 100 trees for bagging. We do not notice significant improvements by using more iterations of bagging. For Additive Groves, the number of trees is automatically selected by the algorithm on the validation set. For P-LS and P-IRLS, we use an R package called "mgcv" [68]. We perform

[2]Random forests is a very competitive full complexity model [20].

5-fold cross validation for all experiments.[3]

3.4 Results

The regression and classification results are presented in Table 3.5 and Table 3.6, respectively. We report means and standard deviations on the 5-fold cross validation test-sets. To facilitate comparison across multiple datasets, we compute normalized scores that average the performance of each method across the datasets, normalized by the accuracy of P-LS/P-IRLS on each dataset.

Table 3.5 and Table 3.6 are laid out as follows: The top of each table shows results for linear/logistic regression (no feature shaping) and the traditional spline-based GAM models P-LS/P-IRLS, BST-SP, and BF-SP. The middle of the tables present results for new methods that do feature shaping with trees instead of splines such as boosted size-limited trees (e.g., BST-TR3), boosted-bagged size-limited trees (e.g., BST-bagTR3), backfitting of boosted trees (e.g., BF-bstTR3), and backfitting of boosted-bagged trees (e.g., BF-bbTR3). The bottom of each table presents results for unrestricted full-complexity models such as random forests and additive groves. Our goal is to devise more powerful GAM models that are as close in accuracy as possible to the full-complexity models, while preserving the intelligibility of linear models.

Several clear patterns emerge in both tables.

(1) There is a large gap in accuracy between linear methods that do not do feature shaping (linear or logistic regression) and most methods that perform feature shaping. For example, on average the spline-based P-LS GAM model has 60% lower normalized RMSE than vanilla linear regression. Similarly, on

[3]We use 5-fold instead of 10-fold cross validation because some of the experiments are very expensive.

average, P-IRLS is about 20% more accurate than logistic regression.

(2) The new tree-based shaping methods are more accurate than the spline-based methods as long as model complexity (and variance — see Section 3.5.1) is controlled. In both tables, the most accurate tree-based GAM models use boosted-bagged trees that are size-limited to 2-4 leaves.

(3) Unrestricted full-complexity models such as random forests and additive groves are more accurate than any of the GAM models because they are able to model feature interactions, which linear models of shaped features cannot capture. Our goal is to push the accuracy of linear shaped models as close as possible to the accuracy of these unrestricted full-complexity models.

Looking more closely at the results for models that shape features with trees, the most accurate model on average is BST-bagTR2 for regression, and BST-bagTR3 for classification. Models that use more leaves are consistently less accurate than comparable models with 2-4 leaves. It is critical to control tree complexity when boosting trees for feature shaping. Moreover, the most accurate methods used bagging inside of boosting to reduce variance. (More on model variance in Section 3.5.1.) Finally, on the regression problems, methods based on gradient boosting of residuals slightly edged out the methods based on backfitting, though the difference is not statistically significant. On the classification problems, however, where backfitting is performed on pseudo-residuals, there were stability problems that caused some runs to diverge or fail to terminate. Overall, tree-based shaping methods based on gradient-boosting appear to be preferable to tree-based methods based on backfitting because the gradient boosting methods may be a little more accurate, are often faster, and on some problems are more robust.

Although tree-based feature shaping yields significant improvements in ac-

curacy for GAMs, on most problems they are not able to close the gap with unrestricted full-complexity models such as random forests. For example, all linear methods have much worse RMSE on the wine regression problem than the unrestricted random forest model. On problems where feature interaction is important, linear models without interaction terms must be less accurate.

3.4.1 Model Selection

There is a risk when comparing many parameterizations of a new method against a small number of baseline methods, that the new method will appear to be better because selecting the best model on the test set leads to overfitting to the test sets. To avoid this, the table includes results for a method called "BST-bagTRX" that uses the cross-validation validation sets (*not* the CV test sets) to pick the best parameters from the BST-bagTRx models for each dataset. This method is not biased by looking at results on test sets, is fully automatic and thus does not depend on human judgement, and is able to select different parameters for each problem. The results in Table 3.5 and Table 3.6 suggest that BST-bagTRX is more accurate than any single fixed parameterization. Looking at the models selected by BST-bagTRX, we see that BST-bagTRX usually picks models with 2, 3 or 4 leaves, and that the model it selects often is the one with the best test-set performance. On both the regression and classification datasets, BF-bagTRX is significantly more accurate than any of the models that use splines for feature shaping.

Table 3.5: RMSE for regression datasets. Each cell contains the mean RMSE ± one standard deviation. Average normalized score on five datasets (excludes synthetic) is shown in the last column, where the score is calculated as relative improvement over P-LS.

Model	Concrete	Wine	Delta	CompAct	Music	Synthetic	Mean
Linear Regression	10.43±0.49	7.55±0.13	5.68±0.14	9.72±0.55	9.61±0.09	1.01±0.00	1.68±0.98
P-LS	5.67±0.41	7.25±0.21	5.67±0.16	2.81±0.13	9.27±0.07	0.04±0.00	1.00±0.00
BST-SP	5.79±0.37	7.27±0.18	5.68±0.18	3.19±0.37	9.29±0.08	0.04±0.00	1.03±0.06
BF-SP	5.66±0.42	7.25±0.21	5.67±0.17	2.77±0.06	9.27±0.08	0.04±0.00	1.00±0.01
BST-TR2	5.19±0.39	7.17±0.10	5.75±0.18	2.68±0.33	9.55±0.08	0.11±0.00	0.98±0.08
BST-TR3	5.13±0.37	7.20±0.16	5.82±0.19	3.18±0.45	9.77±0.07	0.05±0.01	1.02±0.11
BST-TR4	5.24±0.39	7.24±0.15	5.83±0.21	3.70±0.52	9.88±0.09	0.07±0.01	1.07±0.15
BST-TR8	5.57±0.61	7.35±0.17	5.97±0.22	5.07±0.54	10.03±0.10	0.19±0.02	1.19±0.33
BST-TR12	5.92±0.63	7.39±0.13	6.03±0.19	6.59±0.71	10.15±0.07	0.26±0.02	1.31±0.54
BST-TR16	6.08±0.37	7.41±0.23	6.09±0.19	7.07±1.01	10.23±0.08	0.33±0.03	1.36±0.60
BST-bagTR2	5.06±0.39	7.05±0.11	5.67±0.20	**2.59±0.34**	**9.42±0.08**	0.07±0.00	0.96±0.08
BST-bagTR3	4.93±0.41	7.01±0.10	5.67±0.20	2.82±0.35	9.45±0.07	**0.03±0.00**	0.97±0.09
BST-bagTR4	4.99±0.43	7.01±0.12	5.70±0.20	2.95±0.35	9.46±0.08	**0.03±0.00**	0.99±0.09
BST-bagTR8	5.04±0.43	7.04±0.13	5.79±0.18	3.40±0.34	9.48±0.08	0.06±0.00	1.02±0.13
BST-bagTR12	5.11±0.44	7.07±0.15	5.85±0.18	3.76±0.33	9.50±0.07	0.07±0.00	1.05±0.16
BST-bagTR16	5.18±0.49	7.10±0.18	5.91±0.20	4.16±0.39	9.52±0.09	0.09±0.00	1.09±0.21
BST-bagTRX	**4.89±0.37**	**7.00±0.10**	5.65±0.20	**2.59±0.34**	**9.42±0.08**	**0.03±0.00**	**0.95±0.09**
BF-TR	5.80±0.60	7.19±0.09	5.67±0.21	2.81±0.25	9.88±0.08	0.06±0.03	1.02±0.07
BF-bagTR	5.10±0.49	7.02±0.13	**5.61±0.21**	2.69±0.31	9.43±0.07	0.04±0.00	0.97±0.07
BF-bstTR2	5.11±0.37	7.14±0.11	5.73±0.20	2.66±0.35	9.62±0.07	0.13±0.01	0.98±0.08
BF-bstTR3	5.21±0.38	7.29±0.19	5.84±0.21	4.38±0.24	10.77±0.10	0.04±0.01	1.14±0.24
BF-bstTR4	5.49±0.72	7.44±0.20	5.94±0.21	4.97±0.73	11.24±0.07	0.06±0.03	1.21±0.33
BF-bstTR8	6.74±0.76	7.93±0.32	6.08±0.24	9.18±0.77	12.08±0.07	0.04±0.01	1.59±0.87
BF-bstTR12	7.13±0.68	8.10±0.27	6.15±0.24	11.20±0.72	12.31±0.15	0.08±0.03	1.76±1.16
BF-bstTR16	7.22±0.73	8.33±0.35	6.18±0.24	11.41±0.29	12.59±0.10	0.11±0.08	1.79±1.17
BF-bbTR2	5.13±0.41	7.05±0.12	5.66±0.19	2.59±0.37	9.50±0.07	0.12±0.00	0.97±0.08
BF-bbTR3	5.15±0.44	7.07±0.17	5.74±0.20	2.85±0.33	9.80±0.07	0.04±0.00	0.99±0.09
BF-bbTR4	6.20±0.86	7.12±0.22	5.80±0.23	3.01±0.23	9.83±0.08	0.04±0.00	1.05±0.09
BF-bbTR8	6.33±0.46	7.30±0.21	5.95±0.23	3.72±0.84	9.86±0.11	0.04±0.00	1.11±0.16
BF-bbTR12	6.52±0.56	7.52±0.30	6.01±0.20	4.32±0.94	9.89±0.06	0.04±0.00	1.17±0.23
BF-bbTR16	6.37±0.48	7.63±0.26	6.07±0.24	4.85±0.76	9.91±0.07	0.05±0.01	1.21±0.29
Random Forests	4.98±0.44	6.05±0.23	5.34±0.13	2.45±0.09	9.70±0.07	0.55±0.00	0.88±0.06
Additive Groves	4.25±0.47	6.21±0.20	5.35±0.14	2.23±0.15	9.03±0.05	0.02±0.00	0.86±0.10

3.5 Discussion

3.5.1 Bias-Variance Analysis

The results in Tables 3.5 and 3.6 show that adding feature shaping to linear models significantly improves accuracy on problems of small-medium dimensionality, and feature shaping with tree-based models significantly improves accuracy compared to feature shaping with splines. But why are tree-based methods more accurate for feature shaping than spline-based methods? In this section we show that splines tend to underfit, i.e., have very low variance at the expense of higher bias, but tree-based shaping models can have both low bias

Table 3.6: Error rate for classification datasets. Each cell contains the classification error $\pm$ one standard deviation. Averaged normalized score on all datasets is shown in the last column, where the score is calculated as relative improvement over P-IRLS.

Model	Spambase	Insurance	Magic	Letter	Adult	Physics	Mean
Logistic Regression	7.67±1.03	6.11±0.29	20.99±0.46	27.54±0.27	16.04±0.46	29.24±0.36	1.22±0.23
P-IRLS	6.43±0.77	6.11±0.30	14.53±0.41	17.47±0.24	15.00±0.28	29.04±0.49	1.00±0.00
BST-SP	6.24±0.65	6.07±0.31	14.54±0.31	17.61±0.23	15.02±0.25	28.98±0.43	1.00±0.03
BF-SP	6.37±0.29	6.11±0.29	14.58±0.32	17.52±0.17	15.01±0.28	28.98±0.46	1.00±0.03
BST-TR2	5.22±0.77	5.97±0.38	14.63±0.36	17.40±0.22	14.90±0.26	29.58±0.53	0.97±0.08
BST-TR3	5.09±0.79	5.97±0.38	14.54±0.14	17.29±0.25	14.58±0.33	28.81±0.52	0.95±0.08
BST-TR4	5.11±0.70	5.97±0.38	14.60±0.25	17.44±0.26	14.65±0.35	28.72±0.48	0.96±0.08
BST-TR8	5.39±1.06	5.97±0.38	14.64±0.23	17.44±0.27	14.61±0.34	28.77±0.55	0.96±0.08
BST-TR12	5.61±0.76	5.97±0.38	14.57±0.41	17.45±0.24	14.57±0.36	28.63±0.60	0.97±0.08
BST-TR16	5.93±0.96	5.97±0.38	14.83±0.38	17.47±0.23	14.62±0.32	28.63±0.51	0.98±0.05
BST-bagTR2	5.00±0.65	5.97±0.38	14.47±0.20	17.25±0.22	14.95±0.35	29.32±0.67	0.96±0.09
BST-bagTR3	4.89±1.01	5.97±0.38	14.39±0.13	17.22±0.24	**14.57±0.29**	28.65±0.47	0.94±0.09
BST-bagTR4	4.98±1.07	5.98±0.35	14.40±0.28	17.31±0.23	14.63±0.30	29.05±0.50	0.95±0.09
BST-bagTR8	5.22±1.05	5.99±0.36	14.43±0.33	17.42±0.15	14.68±0.35	28.73±0.64	0.96±0.08
BST-bagTR12	5.48±1.09	6.00±0.36	14.44±0.35	17.45±0.19	14.67±0.39	29.06±0.77	0.97±0.07
BST-bagTR16	5.52±1.01	5.99±0.36	14.45±0.29	17.47±0.23	14.69±0.34	28.77±0.65	0.97±0.07
BST-bagTRX	**4.78±0.82**	**5.95±0.37**	**14.31±0.21**	**17.21±0.23**	14.58±0.28	**28.62±0.49**	**0.94±0.09**
BF-TR	6.41±0.37	6.34±0.27	16.81±0.35	17.36±0.26	14.96±0.28	31.64±0.57	1.05±0.07
BF-bagTR	5.63±0.47	6.34±0.22	15.09±0.48	17.41±0.23	14.95±0.34	29.51±0.46	0.99±0.06
BF-bstTR2	5.39±0.68	6.28±0.18	14.43±0.37	17.44±0.35	14.87±0.21	29.70±0.66	0.98±0.35
BF-bstTR3	6.85±1.48	6.31±0.54	15.11±0.24	17.53±0.18	14.64±0.32	29.90±0.34	1.02±0.06
BF-bstTR4	7.63±0.85	6.40±0.48	15.47±0.26	17.46±0.29	14.66±0.27	29.67±0.80	1.05±0.09
BF-bstTR8	10.20±1.30	6.52±0.54	16.26±0.36	17.47±0.25	14.60±0.36	30.32±0.41	1.13±0.24
BF-bstTR12	12.39±1.04	6.53±0.54	16.95±0.40	17.50±0.25	14.76±0.32	31.08±0.43	1.21±0.35
BF-bstTR16	13.11±1.32	6.55±0.58	17.68±0.56	17.52±0.24	14.79±0.33	31.97±0.37	1.24±0.40
BF-bbTR2	5.48±0.59	6.20±0.26	15.26±0.43	17.86±0.30	14.90±0.31	29.36±0.56	0.99±0.08
BF-bbTR3	5.83±0.76	6.42±0.24	14.64±0.18	17.43±0.34	14.77±0.34	28.64±0.56	0.99±0.06
BF-bbTR4	6.13±0.90	6.48±0.20	14.68±0.24	17.43±0.26	14.74±0.35	28.64±0.50	1.00±0.07
BF-bbTR8	6.48±0.97	6.59±0.26	14.79±0.20	17.51±0.27	14.64±0.33	28.65±0.50	1.01±0.07
BF-bbTR12	7.35±1.24	6.56±0.21	14.90±0.22	17.53±0.18	14.58±0.33	28.88±0.29	1.04±0.10
BF-bbTR16	7.72±1.36	6.56±0.20	15.02±0.40	17.52±0.24	14.58±0.28	29.16±0.38	1.05±0.11
Random Forests	4.48±0.64	5.97±0.41	11.99±0.50	6.23±0.27	14.85±0.25	28.55±0.56	0.80±0.23

and low variance if tree complexity is controlled.

To show why spline models do not perform as well as tree models, and why controlling complexity is so critical with trees, we perform a bias-variance analysis on the regression datasets.[4] As in previous experiments, we randomly select 20% of the points as test sets. We then draw L samples of size $M = 0.64N$ points from the remaining points to keep the training sample size the same as with 5-fold cross validation in previous experiments. We use $L = 10$ trials. The bias-variance decomposition is calculated as follows:

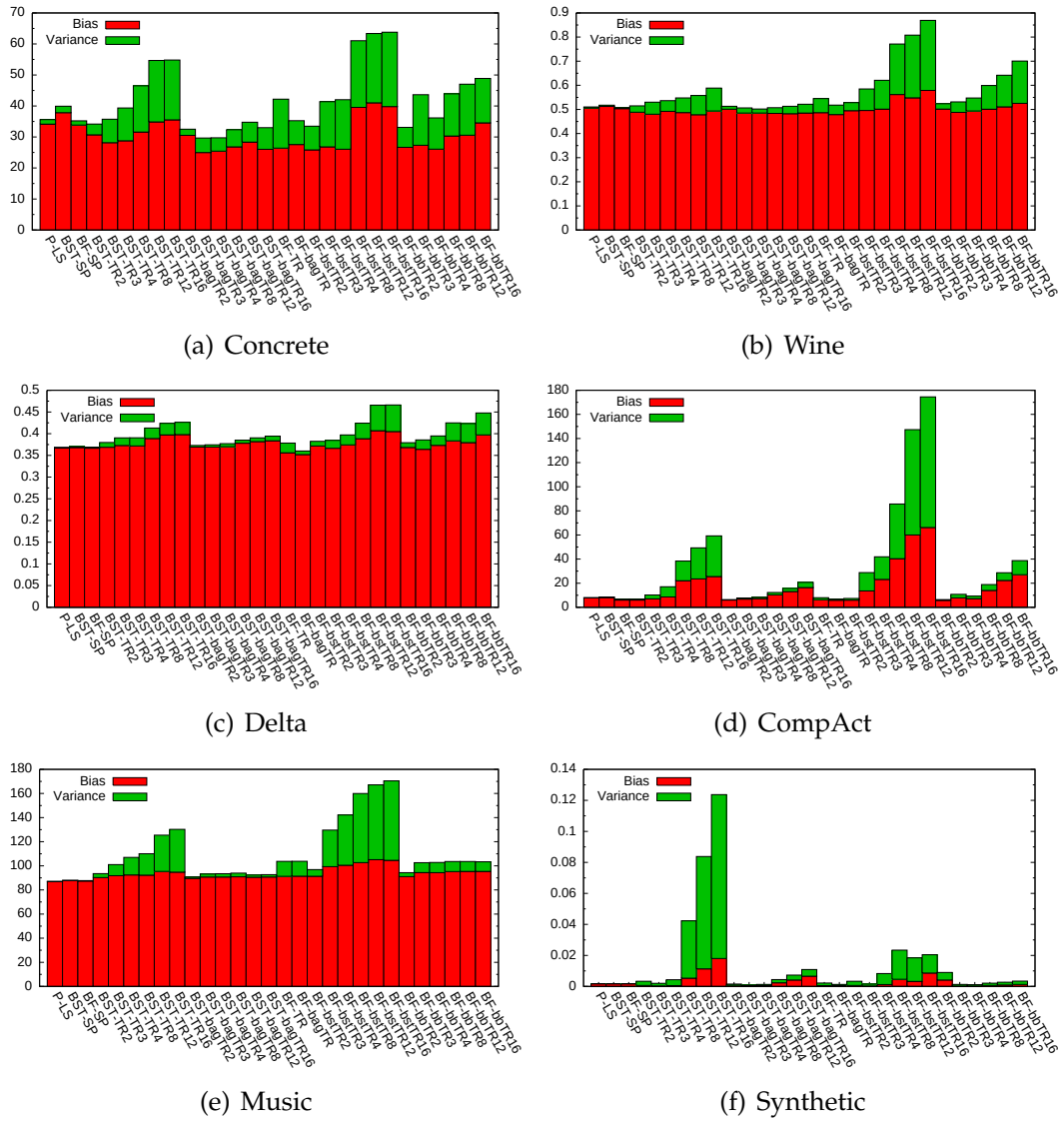
$$Expected\ Loss = (bias)^2 + variance + noise$$

[4]We do not perform bias-variance analysis on the classification problems because the bias-variance decomposition for classification is not as well defined.

Define the average prediction on L samples for each point $(\boldsymbol{x}_i, y_i)$ in test set as $\bar{y}_i = \frac{1}{L}\sum_{l=1}^{L}\hat{y}_i^l$, where $\hat{y}_i^l$ is the predicted value for $\boldsymbol{x}_i$ using sample l. The squared bias $(bias)^2 = \frac{1}{N'}\sum_{i=1}^{N'}[\bar{y}_i - y_i]^2$, where y_i is the known target in the test set and $N' = 0.2N$ is the size of test set. The variance is calculated as $variance = \frac{1}{N'}\sum_{i=1}^{N'}\frac{1}{L}\sum_{l=1}^{L}[\hat{y}_i^l - \bar{y}_i]^2$.

The bias-variance results for the six regression datasets are shown in Figure 3.6. We can see that methods based on regression splines have very low variance, but sometimes at the expense of increased bias, while the best tree-based methods consistently have lower bias combined with low-enough variance to yield better overall RMSE. If tree complexity is not carefully controlled, however, variance explodes and hurts total RMSE. As expected, adding bagging inside boosting further reduces variance, making tree-based feature shaping methods based on gradient-boosting of residuals with internal bagging the most accurate method overall. (We do not expect bagging would help regression splines because the variance of regression splines is so low to begin with.) But even bagging will not prevent overfitting if the trees are too complex. Figure 3.3 shows training curves on the train, validation and test sets for gradient boosting with bagging and backfitting with bagging on a regression and classification problem. BST-bagTR2 is more resistant to overfitting than BST-bagTR16 which easily overfits.

The training curves for backfitting are not monotonic, and have distinct peaks on the classification problem. Each peak corresponds to a new backfitting iteration when pseudo residuals are updated. In our experience, backfitting on classification problems is consistently inferior to other methods, in part because it is harder for the local scoring algorithm to find a "good" set of pseudo residuals, which ultimately leads to instability and poor fit. Interestingly, in the



(a) Concrete (b) Wine

(c) Delta (d) CompAct

(e) Music (f) Synthetic

Figure 3.6: Bias-variance analysis for the six regression problems (bias = red at bottom of bars; variance = green at top of bars).

bias-variance analysis of backfitting, both bias and variance often increase as the trees become larger, and the worst performing models on the five non-synthetic datasets are backfit models with large trees. We suspect backfitting can get stuck in inferior local minima when shaping with larger trees, hurting both bias and variance, which may explain the instability and convergence problems observed with backfitting on some classification problems.

3.5.2 Underfitting, Intelligibility, and Fidelity

One of the main reasons to use GAMs (linear models of non-linearly shaped features) is intelligibility. Figure 3.1 in Section 3.1 showed shape models for features in the synthetic dataset. In this section we show shape models learned for features from real regression and classification problems.

Figure 3.4 shows feature shape plots for the "Concrete" regression problem. Figure 3.5 show shape plots for the "Spambase" clasification problem. In each figure, the top row of plots are from the P-LS spline method, and the bottom row of plots are from BST-bagTR3. Confidence intervals for the least squres method can be computed analytically. Confidence intervals for BST-bagTR3 are generated by running BST-bagTR3 multiple times on bootstrap samples. As expected, the spline-based approach produces smooth plots. The cost of this smoothness, however, is poorer fit that results in higher RMSE or lower accuracy. Moreover, not all phenomena are smooth. Freezing and boiling occur at distinct temperatures, the onset of instability occurs abruptly as fluid flow increases relative to Reynolds Number, and many human decision making processes such as loans, admission to school, or administering medical procedures use discrete thresholds.

On the "Concrete" dataset, all features in Figure 3.4 are clearly non-linear. On this dataset P-LS is sacrificing accuracy for smoothness — the tree-based fitting methods have significantly lower RMSE than P-LS. The smoother P-LS models may appear more appealing and easier to interpret, but there is structure in the BST-bagTR3 models that is less apparent or missing in the P-LS plots that might be informative or important. As just one example, the P-LS and BST-bagTR3 models do not agree on the slopes of parts of the models for the Coarse and Fine Aggregate features.

On the "Spambase" dataset, the shape functions are nonlinear with sharp turns. Again, the BST-bagTR3 model is significantly more accurate than P-IRLS. Interestingly, the spline models for features Freq_hp, Freq_!, Freq_remove and Freq_$ show strong positive or negative slope in the right-hand side of the shape plots where data is sparse (albeit with very wide confidence intervals) while the BST-bagTR3 shape plots appear to be better behaved.

Below each shape plot in Figures 3.4 and 3.5 is the weight of each shape term in the linear model. These weights tell users how important each term is to the model. Terms can be sorted by weight, and if necessary terms with low weight can be removed from the model and the retained features reshaped.

In both figures there is coarse similarity between the feature shape plots learned by the spline and tree-based methods, but in many plots the tree-based methods appear to have caught structure that is missing or more difficult to see in the spline plots. The spline models may be more appealing to the eye, but they are clearly less accurate and appear to miss some of details of the shape functions.

3.5.3 Computational Cost

In our experiments, P-LS and P-IRLS are very fast on small datasets, but on the larger datasets they are slower than the BST-TRx. Due to the extra cost of bagging, BST-bagTRx, BF-bagTR and BF-bbTRx are much slower than P-LS/P-IRLS or BST-TRx. The slowest method we tested is backfitting, which is expensive because at each iteration the previous shape functions are discarded and a new fit for each feature must be learned. Gradient boosting converges faster because in each iteration the algorithm adds a patch to the existing pool of predictors, thus building on previous efforts rather than discarding them.

Gradient boosting is easier to parallelize [54] than backfitting (Gauss-Seidel). The Jacobi method is sometimes used as an alternative to Gauss-Seidel because it is easier to parallelize, however, in our experience, Jacobi-based backfitting converges to suboptimal solutions that can be much worse.

3.5.4 Limitations and Extensions

The experiments in this chapter are on datasets of low-to-medium dimensionality (less than 100 dimensions). Our next step is to scale the algorithm to datasets with more dimensions (and more training points). Even linear models lose intelligibility when there are hundreds of terms in the model. To help retain intelligibility in high dimensional spaces, we have begun developing an extension to BST-bagTRX that incorporates feature selection in the feature shaping process to retain only those features that, after shaping, make the largest contributions to the model. We do not present results for feature selection in this chapter because it is important to focus first on the foundational issue of what algorithm(s) train the best models. We will see results for feature selection in Chapter 5.

In this chapter we focus exclusively on shape functions of individual features; feature interaction is not allowed. Because of this, the models will not be able to achieve the same accuracy as unrestricted full-complexity models on many datasets. The addition of a few carefully selected interaction terms would further close this gap [35]. Because 3-D plots can be visualized, we may be able to allow pairwise interactions in our models while preserving some of the intelligibility. Feature interaction is discussed in Chapter 4.

Our empirical results suggest that bagging small trees of only 2-4 leaves yields the best accuracy. These are very small trees trained for one feature at a time and thus they divide the number line into 2-4 subintervals. We could

imagine replacing bagged decision trees with some type of dynamic programming algorithm that directly works on (possibly smoothed) subintervals of the number line.

3.6 Related Work

Generalized additive models were introduced by the statistics community [32, 68] and have been extended to include LASSO feature selection [58] and to incorporate interaction terms [35]. Binder and Tutz performed a comprehensive comparison of methods for fitting GAMs with regression splines [13]. They compared backfitting, boosting, and penalized iteratively re-weighted least squares on simulated datasets. Our work differs from theirs in that we examine both regression splines and regression trees, most of our experiments are with real datasets, we look at both regression and classification, and we introduce a new method that is more accurate than splines.

Methods have been proposed for fitting GAMs with arbitrary link functions where the link function also is unknown and must be fitted. ACE [16] is probably the most well-known method for fitting these kind of GAMs. We do not evaluate ACE in this work because learned link functions can be complex, making it difficult to interpret the feature shape models. We focus on the identity and logit link functions because these are the link functions appropriate for regression and classification.

Forman *et al.* proposed feature shaping for linear SVM classifiers [24]. Their focus is on estimating the posterior probability $P(y = 1|\boldsymbol{x}_i = v)$.

Recently there have been efforts to scale GAMs. [54] uses MapReduce to parallelize gradient boosting and large tree construction. [64] parallelizes growing regression trees via gradient boosting using a master-worker paradigm where

data are partitioned among workers. The algorithm carefully orchestrates overlap between communication and computation to achieve good performance.

3.7 Conclusions

We present a comprehensive empirical study of algorithms for fitting generalized additive models (GAMs) with spline and tree-based shape functions. Our bias-variance analysis shows that spline-based methods tend to underfit and thus may miss important non-smooth structure in the shape models. As expected, the bias-variance analysis also shows that tree-based methods are prone to overfitting and require careful regularization. We also introduce a new GAM method based on gradient boosting of size-limited bagged trees that yields significantly more accuracy than previous algorithms on both regression and classification problems while retaining the intelligibility of GAM models.

CHAPTER 4

ACCURATE INTELLIGIBLE MODELS WITH PAIRWISE INTERACTIONS

Two heads are better than one.

–John Heywood

4.1 Introduction

Many machine learning techniques such as boosted or bagged trees, SVMs with RBF kernels, or deep neural nets are powerful classification and regression models for high-dimensional prediction problems. However, due to their complexity, the resulting models are hard to interpret for the user. But in many applications, intelligibility is as important as accuracy [46], and thus building models that users can understand is a crucial requirement.

Generalized additive models (GAMs) are the gold standard for intelligibility when only univariate terms are considered [32, 46]. Standard GAMs have the form

$$g(E[y]) = \sum f_i(x_i), \tag{4.1}$$

where g is the link function. Standard GAMs are easy to interpret since users can visualize the relationship between the univariate terms of the GAM and the dependent variable through a plot $f_i(x_i)\ vs.\ x_i$. However there is unfortunately a significant gap between the performance of the best standard GAMs and full complexity models [46]. In particular, Equation 4.1 does not model any interactions between features, and it is this limitation that lies at the core of the lack of accuracy of standard GAMs as compared to full complexity models.

Example 3. *Consider the function* $F(\boldsymbol{x}) = \log(x_1^2 x_3) + x_2 x_3$. *$F$ has a pairwise interaction* (x_2, x_3)*, but no interactions between* (x_1, x_2) *or* (x_1, x_3)*, since* $\log(x_1^2 x_3) = 2\log(x_1) + \log(x_3)$*, which is additive.*

Our first contribution in this chapter is to build models that are more powerful than GAMs, but are still intelligible. We observe that two-dimensional interactions can still be rendered as heatmaps of $f_{ij}(x_i, x_j)$ on the two-dimensional x_i, x_j-plane, and thus a model that includes only one- and two-dimensional components is still intelligible. Therefore in this chapter, we propose building models of the form

$$g(E[y]) = \sum f_i(x_i) + \sum f_{ij}(x_i, x_j); \tag{4.2}$$

we call the resulting model class *Generalized Additive Models plus Interactions*, or short GA^{2}Ms.

The main challenge in building GA^{2}Ms is the large number of pairs of features to consider. We thus only want to include "true" interactions that pass some statistical test. To this end, we focus on problems with up to thousands of features since for truly high dimensional problems (e.g., millions of features), it is almost intractable to test all possible pairwise interactions (e.g., trillions of feature pairs).

Existing approaches for detecting statistical interactions can be divided into two classes. One class of methods directly models and compares the interaction effects and additive effects [26, 29, 45, 68]. One drawback of these methods is that spurious interactions may be reported over low-density regions [35]. The second class of methods measures the performance drop in the model if certain interaction is not included; they compare the performance between restricted and unrestricted models, where restricted models are not allowed to model an interaction in question [60]. Although this class of methods does not suffer from

the problem of low-density regions, they are computationally extremely expensive even for pairwise interaction detection.

Our second contribution in this chapter is to scale the construction of GA^{2}Ms by proposing a novel, extremely efficient method called FAST to measure and rank the strength of the interaction of all pairs of variables. Our experiments show that FAST can efficiently rank all pairwise interactions close to a ground truth ranking.

Our third contribution is an extensive empirical evaluation of GA^{2}M-models. Surprisingly, on many of the datasets included in our study, the performance of GA^{2}M-models is close and sometimes better than the performance of full-complexity models. These results indicate that GA^{2}M-models not only make a significant step in improving accuracy over standard GAMs, but in some cases they actually come all the way to the performance of full-complexity models. The performance may be due to the difficulty of estimating intrinsically high dimensional functions from limited data, suggesting that the bias associated with the GA^{2}M structure is outweighed by a drop in variance. We also demonstrate that the resulting models are intelligible through a case study.

In this chapter we make the following contributions:

- We introduce the model class GA^{2}M.
- We introduce our new method FAST for efficient interaction detection. (Section 4.4)
- We show through an extensive experimental evaluation that (1) GA^{2}Ms have accuracy comparable to full-complexity models; (2) FAST accurately ranks interactions as compared to a gold standard; and (3) FAST is computationally efficient. (Section 4.5)

We start with a problem definition and a survey of related work in Sections 4.2

and 4.3.

4.2 Problem Definition

Let $\mathcal{D} = \{(\boldsymbol{x}_i, y_i)\}_1^N$ denote a dataset of size N, where $\boldsymbol{x}_i = (x_{i1}, ..., x_{in})$ is a feature vector with n features and y_i is the response. Let $\boldsymbol{x} = (x_1, ..., x_n)$ denote the variables or features in the dataset. For $u \subseteq \{1, ..., n\}$, we denote by x_u the subset of variables whose indices are in u. Similarly x_{-u} will indicate the variables with indices not in u. To simplify notation, we denote $\mathcal{U}^1 = \{\{i\}|1 \leq i \leq n\}$, $\mathcal{U}^2 = \{\{i, j\}|1 \leq i < j \leq n\}$, and $\mathcal{U} = \mathcal{U}^1 \cup \mathcal{U}^2$, i.e., $\mathcal{U}$ contains all indices for all features and pairs of features.

For any $u \in \mathcal{U}$, let $\mathcal{H}_u$ denote the Hilbert space of Lebesgue measurable functions $f_u(x_u)$, such that $E[f_u] = 0$ and $E[f_u^2] < \infty$, equipped with the inner product $\langle f_u, f'_u \rangle = E[f_u f'_u]$. Let $\mathcal{H}^1 = \sum_{u \in \mathcal{U}^1} \mathcal{H}_u$ denote the Hilbert space of functions that have additive form $F(\boldsymbol{x}) = \sum_{u \in \mathcal{U}^1} f_u(x_u)$ on univariate compnents; we call those components *shape functions* [46]. Similarly let $\mathcal{H} = \sum_{u \in \mathcal{U}} \mathcal{H}_u$ denote the Hilbert space of functions of $\boldsymbol{x} = (x_1, ..., x_n)$ that have additive form $F(\boldsymbol{x}) = \sum_{u \in \mathcal{U}} f_u(x_u)$ on both one- and two-dimensional shape functions. Models described by sums of low-order components are called *generalized additive models (GAMs)*, and in the remainder of the paper, we use GAMs to denote models that only consist of univariate terms.

We want to find the best model $F \in \mathcal{H}$ that minimizes the following objective function:

$$\min_{F \in \mathcal{H}} E[L(y, F(\boldsymbol{x}))], \tag{4.3}$$

where $L(\cdot, \cdot)$ is a non-negative convex loss function. When L is the squared loss, our problem becomes a regression problem, and if L is logistic loss function, we

are dealing with a classification problem.

4.3 Existing Approaches

4.3.1 Fitting Generalized Additive Models

Terms in GAMs can be represented by a variety of functions, including splines [67], regression trees, or tree ensembles [10]. There are two popular methods of fitting GAMs: Backfitting [32] and gradient boosting [26]. When the shape function is spline, fitting GAMs reduces to fitting generalized linear models with different bases, which can be solved by least squares or iteratively reweighted least squares [68].

Spline-based methods become inefficient when modeling higher order interactions because the number of parameters to estimate grows exponentially; tree-based methods are more suitable in this case. Standard additive modeling only involves modeling individual features (also called *feature shaping*). Previous research showed that gradient boosting with ensembles of shallow regression trees is the most accurate method among a number of alternatives [46].

4.3.2 Interaction Detection

In this section, we briefly review existing approaches to interaction detection.

ANOVA. An additive model is fit with all pairwise interaction terms [32] and the significance of interaction terms is measured through an analysis of variance (ANOVA) test [68]. The corresponding p-value for each pair can then be computed; however, this requires the computation of the full model, which is prohibitively expensive.

Partial Dependence Function. Friedman and Popescu proposed the following statistic to measure the strength of pairwise interactions,

$$H_{ij}^2 = \frac{\sum_{k=1}^{N}[\hat{F}_{ij}(x_{ki}, x_{kj}) - \hat{F}_i(x_{ki}) - \hat{F}_j(x_{kj}))]^2}{\sum_{k=1}^{N} \hat{F}_{ij}^2(x_{ki}, x_{kj})} \tag{4.4}$$

where $\hat{F}_u(x_u) = E_{x_{-u}}[F(x_u, x_{-u})]$ is the partial dependence function (PDF) [26, 29] and F is a complex multi-dimensional function learned on the dataset. Computing $\hat{F}_u(x_u)$ on the whole dataset is expensive, thus one often specifies a subset of size m on which to compute $\hat{F}_u(x_u)$. The complexity is then $O(m^2)$. However, since partial dependence functions are computed based on uniform sampling, they may detect spurious interactions over low-density regions [35].

GUIDE. GUIDE tests pairwise interactions based on the χ^2 test [45]. An additive model F is fit in $\mathcal{H}^1$ and residuals are obtained. To detect interactions for (x_i, x_j), GUIDE divides the (x_i, x_j)-space into four quadrants by splitting the range of each variable into two halves at the sample median. Then GUIDE constructs a 2×4 contingency table using the residual signs as rows and the quadrants as columns. The cell values in the table are the number of "+"s and "-"s in each quadrant. These counts permit the computation of a p-value to measure the interaction strength of a pair. While this might be more robust to outliers, in practice it is less powerful than the method we propose.

Grove. Sorokina *et al.* proposed a grove-based method to detect statistical interactions [60]. To measure the strength of a pair (x_i, x_j), they build both the restricted model $R_{ij}(\boldsymbol{x})$ and unrestricted model $F(\boldsymbol{x})$, where $R_{ij}(\boldsymbol{x})$ is prevented from modeling an interaction (x_i, x_j):

$$R_{ij}(\boldsymbol{x}) = f_{\setminus i}(x_1, ..., x_{i-1}, x_{i+1}, ..., x_n) + f_{\setminus j}(x_1, ..., x_{j-1}, x_{j+1}, ..., x_n). \tag{4.5}$$

To correctly estimate interaction strength, such method requires a model to be highly predictive when certain interaction is not allowed to appear, and there-

fore many learning algorithms are not applicable (e.g., bagged decision trees). To this end, they choose to use Additive Groves [59].

They measure the performance as standardized root mean squared error (RMSE) and quantify the interaction strength I_{ij} by the difference between $R_{ij}(\boldsymbol{x})$ and $F(\boldsymbol{x})$,

$$stRMSE(F(\boldsymbol{x})) = \frac{RMSE(F(\boldsymbol{x}))}{StD(F^*(\boldsymbol{x}))} \tag{4.6}$$

$$I_{ij} = stRMSE(R_{ij}(\boldsymbol{x})) - stRMSE(F(\boldsymbol{x})) \tag{4.7}$$

where $Std(F^*(\boldsymbol{x}))$ is calculated as standard deviation of the response values in the training set. The ranking of all pairs can be generated based on the strength I_{ij}.

To handle correlations among features, they use a variant of backward elimination [30] to do feature selection. Although Grove is accurate in practice, building restricted and unrestricted models are computationally expensive and therefore this method is almost infeasible for large high dimensional datasets.

4.4 Our Approach

For simplicity and without loss of generality, we focus in this exposition on regression problems. Since there are $O(n^2)$ pairwise interactions, it is very hard to detect pairwise interactions when n is large. Therefore we propose a framework using greedy forward stagewise selection strategy to build the most accurate model in $\mathcal{H}$.

Algorithm 9 summarizes our approach called GA^{2}M. We maintain two sets $\mathcal{S}$ and $\mathcal{Z}$, where $\mathcal{S}$ contains the selected pairs so far and $\mathcal{Z}$ is the set of the remaining pairs (Line 1-2). We start with the best additive model F so far in Hilbert

Algorithm 9 GA^{2}M Framework

1: $\mathcal{S} \leftarrow \varnothing$
2: $\mathcal{Z} \leftarrow \mathcal{U}^2$
3: **while** not converge **do**
4: $\quad F \leftarrow \arg\min_{F \in \mathcal{H}^1 + \sum_{u \in \mathcal{S}} \mathcal{H}_u} \frac{1}{2} E[(y - F(\boldsymbol{x}))^2]$
5: $\quad R \leftarrow y - F(\boldsymbol{x})$
6: $\quad$ **for all** $u \in \mathcal{Z}$ **do**
7: $\quad\quad F_u \leftarrow E[R|x_u]$
8: $\quad u^* \leftarrow \arg\min_{u \in \mathcal{Z}} \frac{1}{2} E[(R - F_u(x_u))^2]$
9: $\quad \mathcal{S} \leftarrow \mathcal{S} \cup \{u^*\}$
10: $\quad \mathcal{Z} \leftarrow \mathcal{Z} - \{u^*\}$

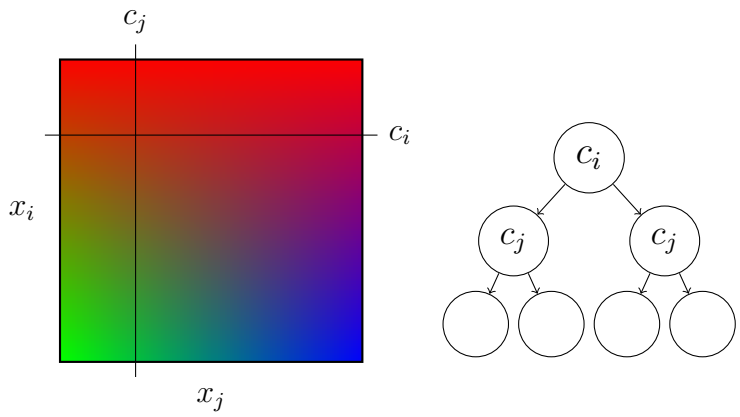


Figure 4.1: Illustration for searching cuts on input space of x_i and x_j. On the left we show a heat map on the target for different values of x_i and x_j. c_i and c_j are cuts for x_i and x_j, respectively. On the right we show an extremely simple predictor of modeling pairwise interaction.

space $\mathcal{H}^1 + \sum_{u \in \mathcal{S}} \mathcal{H}_u$ (Line 4) and detect interactions on the residual R (Line 5). Then for each pair in $\mathcal{Z}$, we build an interaction model on the residual R (Line 6-7). We select the best interaction pair and include it in $\mathcal{S}$ (Line 9-10). We then repeat this process until there is no gain in accuracy.

Note that Algorithm 9 will find an overcomplete set $\mathcal{S}$ by the greedy nature of the forward selection strategy. When features are correlated, it is also possible that the algorithm includes false pairs. For example, consider the function in Example 3. If x_1 is highly correlated with x_3, then (x_1, x_2) may look like an interaction pair, and it may be included in $\mathcal{S}$ before we select (x_2, x_3). But since

we will refit the model every time we include a new pair, it is expected that F will perfectly model (x_2, x_3) and therefore (x_1, x_2) will become a less important term in F.

For large high-dimensional datasets, however, Algorithm 9 is very expensive for two reasons. First, fitting interaction models for $O(n^2)$ pairs in $\mathcal{Z}$ can be very expensive if the model is non-trivial. Second, every time we add a pair, we need to refit the whole model, which is also very expensive for large datasets. As we will see in Section 4.4.1 and Section 4.4.2, we will relax some of the constraints in Algorithm 9 to achieve better scalability while still staying accurate.

4.4.1 Fast Interaction Detection

Consider the conceptual additive model in Equation 4.2, given a pair of variables (x_i, x_j) we wish to measure how much benefit we can get if we model $f_{ij}(x_i, x_j)$ instead of $f_i(x_i) + f_j(x_j)$. Since we start with shaping individual features and always detect interactions on the residual, $f_i(x_i) + f_j(x_j)$ are presumably modeled and therefore we only need to look at the residual sum of squares (RSS) for the interaction model f_{ij}. The intuition is that when (x_i, x_j) is a strong interaction, modeling f_{ij} can significantly reduce the RSS. However, we do not wish to fully build f_{ij} since this is a very expensive operation; instead we are looking for a cheap substitute.

Overview

Our idea is to build an extremely simple model for f_{ij} using cuts on the input space of x_i and x_j, as illustrated in Figure 4.1. The simplest model we can build is to place one cut on each variable, i.e., we place one c_i and one cut c_j on x_i and x_j, respectively. Those cuts are parallel to the axes. The interaction predictor T_{ij}

is constructed by taking the mean of all points in each quadrant. We search for all possible (c_i, c_j) and pick the best T_{ij} with the lowest RSS, which is assigned as weight for (x_i, x_j) to measure the strength of interaction.

Constructing Predictors

Naïve implementation of FAST is straightforward, but careless implementation has very high complexity since we need to repeatedly build a lot of T_{ij} for different cuts. The key insight for faster version of FAST is that we do not need to scan through the dataset each time to compute T_{ij} and compute its RSS. We show that by using very simple bookkeeping data structures, we can greatly reduce the complexity.

Let $\text{dom}(x_i) = \{v_i^1, ..., v_i^{d_i}\}$ be a sorted set of possible values for variable x_i, where $d_i = |\text{dom}(x_i)|$. Define $H_i^t(v)$ as the sum of targets when $x_i = v$, and define $H_i^w(v)$ as the sum of weights (or counts) when $x_i = v$. Intuitively, these are the standard histograms when constructing regression trees. Similarly, we define $CH_i^t(v)$ and $CH_i^w(v)$ as the cumulative histogram for sum of targets and sum of weights, respectively, i.e., $CH_i^t(v) = \sum_{u \le v} H_i^t(u)$ and $CH_i^w(v) = \sum_{u \le v} H_i^w(u)$. Accordingly, define $\overline{CH_i^t}(v) = \sum_{u > v} H_i^t(u) = CH_i^t(v_i^{d_i}) - CH_i^t(v)$ and define $\overline{CH_i^w}(v) = \sum_{u > v} H_i^w(u) = CH_i^w(v_i^{d_i}) - CH_i^w(v)$. Furthermore, define $H_{ij}^t(u, v)$ and $H_{ij}^w(u, v)$ as the sum of targets and the sum of weights, respectively, when $(x_i, x_j) = (u, v)$.

Consider again the input space for (x_i, x_j), we need a quick way to compute the sum of targets and sum of weights for each quadrant. Figure 4.2 shows an example for computing sum of targets on each quadrant. Given the above notations, we already know the marginal cumulative histograms for x_i and x_j, but unfortunately using these marginal values only can not recover values on

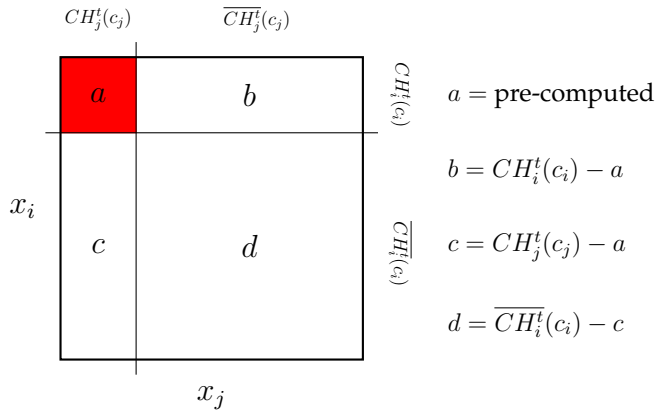


Figure 4.2: Illustration for computing sum of targets for each quadrant. Given that the value of red quadrant is known, we can easily recover values in other quadrant using marginal cumulative histograms.

four quadrants. Thus, we have to compute value for one quadrant.

We show that it is very easy and efficient to compute all possible values for the red quadrant given any cuts (c_i, c_j) using dynamic programming. Once that quadrant is known, we can easily recover values in other quadrant using marginal cumulative histograms. We store those values into lookup tables. Let $L^t(c_i, c_j) = [a, b, c, d]$ be the lookup table for sum of targets on cuts (c_i, c_j), and denote $L^w(c_i, c_j) = [a, b, c, d]$ as the lookup table for sum of weights on cuts (c_i, c_j).

Algorithm 10 describes how to compute the lookup table L^t. We focus on computing quadrant a and other quadrants can be easily computed, which is handled by subroutine $ComputeValues$. Given H_{ij}^t, we first compute as for the first row of L^t (Line 3-5). Let $a[p][q]$ denote the value for cuts (p, q). Note $a[p][q] = a[p-1][q] + \sum_{k \le q} H_{ij}^t(v_i^p, v_j^k)$. Thus we can efficiently compute the rest of the lookup table row by row (Line 6-11).

Once we have L^t and L^w, given any cuts (c_i, c_j), we can easily construct T_{ij}. For example, we can set the leftmost leaf value in T_{ij} as $L^t(c_i, c_j).a / L^w(c_i, c_j).a$. It is easy to see that with those bookkeeping data structures, we can reduce the

Algorithm 10 ConstructLookupTable

$sum \leftarrow 0$
for $q = 1$ to d_j **do**
  $sum \leftarrow sum + H_{ij}^t(v_i^1, v_j^q)$
  $a[1][q] \leftarrow sum$
  $L(v_i^1, v_j^q) \leftarrow ComputeValues(CH_i^t, CH_j^t, a[1][q])$
for $p = 2$ to d_i **do**
  $sum \leftarrow 0$
  for $q = 1$ to d_j **do**
    $sum \leftarrow sum + H_{ij}^t(v_i^p, v_j^q)$
    $a[p][q] \leftarrow sum + a[p-1][q]$
    $L(v_i^p, v_j^q) \leftarrow ComputeValues(CH_i^t, CH_j^t, a[p][q])$

complexity of building predictors to $O(1)$.

Calculating RSS

In this section, we show that calculating RSS for T_{ij} can be very efficient. Consider the definition of RSS. Let $T_{ij}.r$ denote the prediction value on region r, where $r \in \{a, b, c, d\}$.

$$RSS = \sum_{k=1}^{N} (y_k - T_{ij}(\boldsymbol{x}_k))^2 \tag{4.8}$$

$$= \sum_{k=1}^{N} (y_k^2 - 2y_k T_{ij}(\boldsymbol{x}_k) + T_{ij}^2(\boldsymbol{x}_k)) \tag{4.9}$$

$$= \left(\sum_{k=1}^{N} y_k^2 - 2 \sum_{k=1}^{N} y_k T_{ij}(\boldsymbol{x}_k) + \sum_{k=1}^{N} T_{ij}^2(\boldsymbol{x}_k) \right) \tag{4.10}$$

$$= \left(\sum_{k=1}^{N} y_k^2 - 2 \sum_{r} T_{ij}.r \sum_{x_k \in r} y_k + \sum_{r} (T_{ij}.r)^2 L^w.r \right) \tag{4.11}$$

$$= \left(\sum_{k=1}^{N} y_k^2 - 2 \sum_{r} T_{ij}.r L^t.r + \sum_{r} (T_{ij}.r)^2 L^w.r \right) \tag{4.12}$$

In practical implementation, we only need to care about $\sum_r (T_{ij}.r)^2 L^w.r - 2 \sum_r T_{ij}.r L^t.r$ since we are only interested in relative ordering of RSS, and it is easy to see the complexity of computing RSS for T_{ij} is $O(1)$.

Complexity Analysis

For each pair (x_i, x_j), computing the histograms and cumulative histograms needs to scan through the data and therefore its complexity is $O(N)$. Constructing the lookup tables takes $O(d_i d_j + N)$ time. Thus, the time complexity of FAST is $O(d_i d_j + N)$ for one pair (x_i, x_j). Besides, Since we need to store d_i-by-d_j matrices for each pair, the space complexity is $O(d_i d_j)$.

For continuous features, $d_i d_j$ can be quite large. However, we can discretize the features into b equi-frequency bins. Such feature discretizing usually does not hurt the performance of regression tree [43]. As we will see in Section 4.5, FAST is not sensitive to a wide range of bs. Therefore, the complexity can be reduced to $O(b^2 + N)$ per pair when we discretize features into b bins. For small bs ($b \leq 256$), we can quickly process each pair.

4.4.2 Two-stage Construction

With FAST, we can quickly rank of all pairs in $\mathcal{Z}$, the remaining pair set, and add the best interaction to the model. However, refitting the whole model after each pair is added can be very expensive for large high-dimensional datasets. Therefore, we propose a two-stage construction approach.

1. In Stage 1, build the best additive model F in $\mathcal{H}^1$ using only one-dimensional components.
2. In Stage 2, fix the one-dimensional functions, and build models for pairwise interactions on residuals.

Implementation Details

To scale up to large datasets and many features, we discretize the features into 256 equi-frequency bins for continuous features.[1] We find such feature discretization rarely hurts the performance but substantially reduces the running time and memory footprint since we can use one byte to store a feature value. Besides, discretizing the features removes the sorting requirement for continuous features when searching for the best cuts in the space.

Previous research showed that feature shaping using gradient boosting [26] with shallow regression tree ensembles can achieve the best accuracy [46]. We follow similar approach (i.e., gradient boosting with shallow tree-like ensembles) in this work. However, a regression tree is not the ideal learning method for each component for two reasons. First, while regression trees are good as a generic shape functions for any x_u, shaping a single feature is equivalent to cutting on a line, but line cutting can be made more efficient than regression tree. Second, using regression tree to shape pairwise functions can be problematic. Recall that in Stage 1, we obtain the best additive model after gradient boosting converges. This means adding more cuts to any one feature does not reduce the error, and equivalently, any cut on a single feature is random. Therefore, when we begin to shape pairwise interactions, the root test in a regression tree that is constructed greedily top-down is random.

Similar to [46], to effectively shape pairwise interactions, we build shallow tree-like models on the residuals as illustrated in Figure 4.3. We enumerate all possible cuts c_i on x_i. Given this cut, we greedily search the best cut c_j^1 in the region above c_i and similarly greedily search the best cut c_j^2 in the region below c_i. Note we can reuse the lookup table L^t and L^w we developed for FAST for

[1]Note that this is not the number of bins used in FAST, the interaction detection process. Here we use 256 bins for feature/pair shaping.

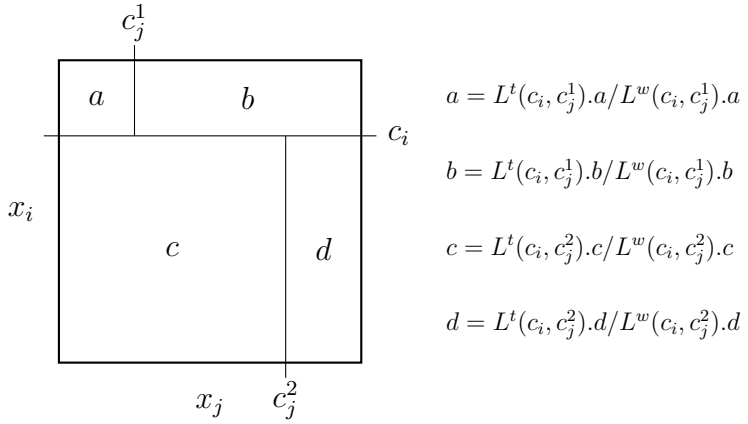


Figure 4.3: Illustration for computing shape function for pairwise interaction.

fast search of those three cuts. Figure 4.3 shows an example of computing the leaf values given c_i, c_j^1 and c_j^2. Similarly, we can quickly compute the RSS given any combination of 3 cuts once the leaf values are available, just as we did in Section 4.4.1, and therefore it is very fast to search for the best combination of cuts in this space. Similarly, we search for the best combination of 3 cuts with 1 cut on x_j and 2 cuts on x_i and pick the better model with lower RSS. It is easy to see the complexity is $O(N + b^2)$, where b is the number of bins for each feature and $b = 256$ in our case.

Further Relaxation

For large datasets, even refitting the model on selected pairs can be very expensive. Therefore, we propose to use the ranking of FAST right after Stage 1, to select the top-K pairs to $\mathcal{S}$, and fit a model using the pairs in $\mathcal{S}$ on the residual R, where K is chosen according to computing power.

Diagnostics

Models that combine both accuracy and intelligibility are important. Usually $\mathcal{S}$ will still be an overcomplete set. For intelligibility, once we have learned the best

Table 4.1: Datasets.

Dataset	Size	Attributes	%Pos
Delta	7192	6	-
CompAct	8192	22	-
Pole	15000	49	-
CalHousing	20640	9	-
MSLR10k	1200192	137	-
Spambase	4601	58	39.40
Gisette	6000	5001	50.00
Magic	19020	11	64.84
Letter	20000	17	49.70
Physics	50000	79	49.72

model in $\mathcal{H}$, we would like to rank all terms (one- and two-dimensional components) so that we can focus on the most important features, or pairwise interactions. Therefore, we need to assign weights for each term. We use $\sqrt{E[f_u^2]}$, the standard deviation of f_u (since $E[f_u] = 0$), as the weight for term u. Note this is a natural generalization of the weights in the linear models; this is easy to see since $f_i(x_i) = w_i x_i$, $\sqrt{E[f_i^2]}$ is equivalent to $|w_i|$ if features are normalized so that $E[x_i^2] = 1$.

4.5 Experiments

In this section we report experimental results on both synthetic and real datasets. The results in Section 4.5.1 show GA^{2}M learns models that are nearly as accurate as full-complexity random forest models while using terms that depend only on single features and pairwise interactions and thus are intelligible. The results in Section 4.5.2 demonstrate that FAST finds the most important interactions of $O(n^2)$ feature pairs to include in the model. Section 4.5.3 compares the computational cost of FAST and GA^{2}M to competing methods. Section 4.5.4 briefly discusses several important design choices made for FAST and GA^{2}M.

Table 4.2: RMSE for regression datasets. Each cell contains the mean RMSE ± one standard deviation. Average normalized score is shown in the last column, calculated as relative improvement over GAM.

Model	Delta	CompAct	Pole	CalHousing	MSLR10k	Mean
Linear Reg.	0.58±0.01	7.92±0.47	30.41±0.24	7.28±0.80	0.76±0.00	1.52±0.79
GAM	0.57±0.02	2.74±0.04	21.62±0.38	5.76±0.55	0.75±0.00	1.00±0.00
GA^{2}M Rand	-	-	11.37±0.38	-	0.73±0.00	-
GA^{2}M Coef	-	-	11.61±0.43	-	0.73±0.00	-
GA^{2}M Order	-	-	10.81±0.29	-	0.74±0.00	-
GA^{2}M FAST	**0.55±0.02**	**2.53±0.02**	**10.59±0.35**	**5.00±0.91**	**0.73±0.00**	**0.84±0.20**
Rand. Forests	0.53±0.19	2.45±0.08	11.38±1.03	4.90±0.81	0.71±0.00	0.83±0.17

Table 4.3: Error rate for classification datasets. Each cell contains the error rate ± one standard deviation. Average normalized score is shown in the last column, calculated as relative improvement over GAM.

Model	Spambase	Gisette	Magic	Letter	Physics	Mean
Logistic Reg.	6.22±0.93	15.78±3.28	17.11±0.08	27.54±0.27	30.02±0.37	1.79±1.25
GAM	5.09±0.64	3.95±0.65	14.85±0.28	17.84±0.20	28.83±0.24	1.00±0.00
GA^{2}M Rand	5.04±0.52	3.53±0.61	-	-	28.82±0.25	-
GA^{2}M Coef	4.89±0.54	3.43±0.55	-	-	28.74±0.37	-
GA^{2}M Order	4.93±0.65	3.08±0.55	-	-	28.76±0.34	-
GA^{2}M FAST	**4.78±0.70**	**2.91±0.38**	**13.88±0.32**	**8.62±0.31**	**28.20±0.18**	**0.81±0.21**
Rand. Forests	4.76±0.70	3.25±0.47	12.45±0.64	6.16±0.22	28.48±0.40	0.79±0.26

Finally, Section 4.5.5 concludes with a case study.

4.5.1 Model Accuracy on Real Datasets

We run experiments on ten real datasets to show the accuracy that GA^{2}M can achieve with models that depend only on 1-d features and pairwise feature interactions.

Datasets

Table 4.1 summarizes the 10 datasets. Five are regression problems: "Delta" is the task of controlling the ailerons of an F16 aircraft [2]. "CompAct" is from the Delve repository and describes the state of multiuser computers [3].

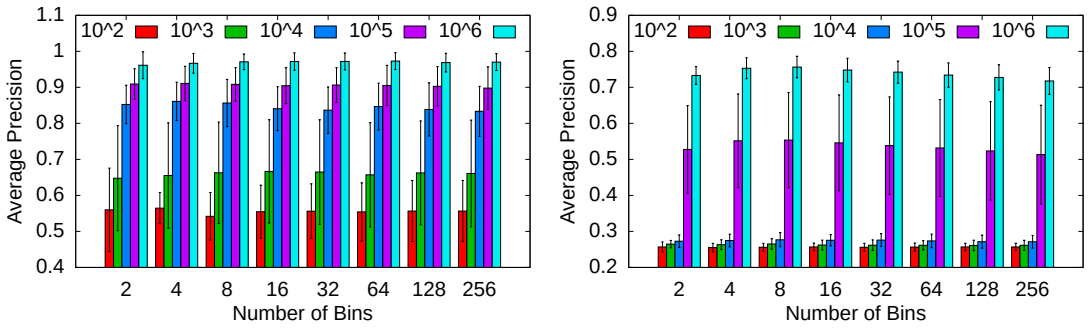
"Pole" describes a telecommunication problem [66]. "CalHousing" describes how housing prices depend on census variables [38]. "MSLR10k" is a learning-to-rank dataset but we treat relevance as regression targets [6]. The other five datasets are binary classification problems: The "Spambase", "Magic" and "Letter" datasets are from the UCI repository [1]. "Gisette" is from the NIPS feature selection challenge [7]. "Physics" is from the KDD Cup 2004 [4].

The features in all datasets are discretized into 256 equi-frequency bins. For each model we include at most 1000 feature pairs; we include all feature pairs in the six problems with least dimension, and the top 1000 feature pairs found by FAST on the "Pole", "MSLR10k", "Spambase", "Gisette", and "Physics" datasets. Although it is possible that higher accuracy might be obtained by including more or fewer feature pairs, search for the optimal number of pairs is expensive and GA^{2}M is reasonably robust to excess feature pairs. However, it is too expensive to include all feature pairs on problems with many features. We use 8 bins for FAST in all experiments.

Results

We compare GA^{2}M to linear/logistic regression, feature shaping (GAMs) without interactions, and full-complexity random forests. For regression problems we report root mean squared error (RMSE) and for classification problems we report $0/1$ loss. To compare results across different datasets, we normalize results by the error of GAMs on each dataset. For all experiments, we train on 80% of the data and hold aside 20% of the data as test sets.

In addition to FAST, we also consider three baseline methods on five high dimensional datasets, i.e., GA^{2}M Rand, GA^{2}M Coef and GA^{2}M Order. GA^{2}M Rand means we add same number of random pairs to GAM. GA^{2}M Or-



(a) 10 features. (b) 100 features.

Figure 4.4: Sensitivity of FAST to the number of bins.

der and GA^{2}M Coef use the weights of 1-d features in GAM to propose pairs; GA^{2}M Order generates pairs by the order of 1-d features and GA^{2}M Coef generates pairs by the product of weights of 1-d features.

The regression and classification results are presented in Table 4.2 and Table 4.3. As expected, the improvement over linear models from shaping individual features (GAMs) is substantial: on average feature shaping reduces RMSE 34% on the regression problems, and reduces 0/1 loss 44% on the classification problems. What is surprising, however, is that by adding shaped pairwise interactions to the models, GA^{2}M FAST substantially closes the accuracy gap between unintelligible full-complexity models such as random forests and GAMs. On some datasets, GA^{2}M FAST even outperforms the best random forest model. Also, none of the baseline methods perform comparably GA^{2}M FAST.

4.5.2 Detecting Feature Interactions with FAST

In this section we evaluate how accurately FAST detects feature interactions on synthetic problems.

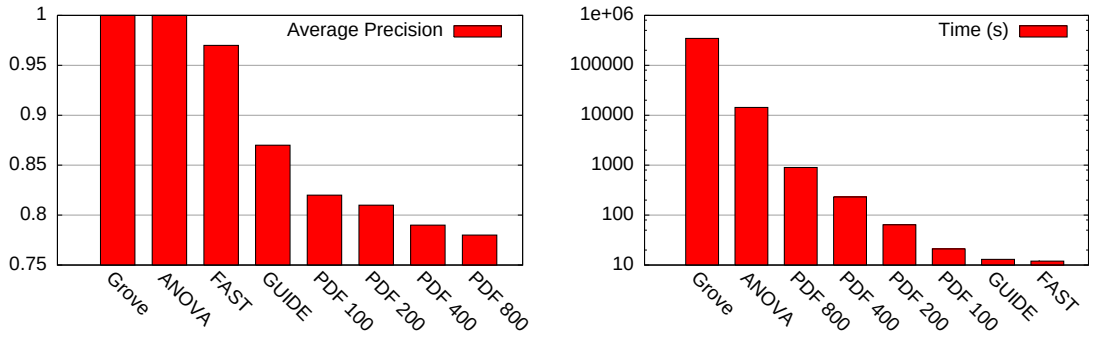


Figure 4.5: Precision/Cost on synthetic function.

Sensitivity to the Number of Bins

To evaluate sensitivity of FAST we use the synthetic function generator in [26] to generate random functions. Because these are synthetic function, we know the ground truth interacting pairs and use average precision (area under the precision-recall curve evaluated at true points) as the evaluation metric. We vary $b = 2, 4, ..., 256$ and the dataset size $N = 10^2, 10^3, ..., 10^6$. For each fixed N, we generate datasets with n features and k higher order interactions x_u, where $|u| = \lfloor 1.5 + r \rfloor$ and r is drawn from an exponential distribution with mean $\lambda = 1$. We experiment with two cases: 10 features with 25 higher order interactions and 100 features with 1000 higher order interactions.

Figure 4.4 shows the mean average precision and variance for 100 trials at each setting. As expected, average precision increases as dataset size increases, and decreases as the number of features increases from 10 (left graph) to 100 (right graph). When there are only 10 features and as many as 10^6 samples, FAST ranks *all* true interactions above all non-interacting pairs (average precision = 1) in most cases, but as the sample size decreases or the problem difficulty increases average precision drops below 1. In the graph on the right with 100 features there are 4950 feature pairs, and FAST needs large sample sizes (10^6

or greater) to achieve average precision above 0.7, and as expected performs poorly when there are fewer samples than pairs of features.

On these test problems the optimal number of bins appears to be about $b = 8$, with average precision falling slightly for number of bins larger and smaller than 8. This is a classic bias-variance tradeoff: smaller b reduces the chances of overfitting but at the risk of failing to model some kinds of interactions, while large b allows more complex interactions to be modeled but at the risk of allowing some false interactions to be confused with weak true interactions.

Accuracy

The previous section showed that FAST accurately detects feature interactions when the number of samples is much larger than the number of feature pairs, but that accuracy drops as the number of feature pairs grows comparable to and then larger than the number of samples. In this section we compare the accuracy of FAST to the interaction detection methods discussed in Section 4.3.2. For ANOVA, we use R package `mgcv` to compute p-values under a Wald test [68]. For PDF, we use `RuleFit` package and we choose $m = 100, 200, 400, 800$, where m is the sample size that trades off efficiency and accuracy [8]. Grove is available in `TreeExtra` package [5].

Here we conduct experiments on synthetic data generated by the following function [34, 60].

$$F(\boldsymbol{x}) \;=\; \pi^{x_1 x_2}\sqrt{2x_3} - \sin^{-1}(x_4) + \log(x_3 + x_5) - \frac{x_9}{x_{10}}\sqrt{\frac{x_7}{x_8}} - x_2 x_7 \qquad (4.13)$$

Variables x_4, x_5, x_8, x_{10} are uniformly distributed in $[0.6, 1]$ and the other variables are uniformly distributed in $[0, 1]$. We generate $10,000$ points for these experiments. Figure 4.5(a) shows the average precision of the methods. On this problem, the Grove and ANOVA methods are accurate and rank all 11 true pairs

in the top of the list. FAST is almost as good and correctly ranks the top ten pairs. The other methods are significantly less accurate than Grove, ANOVA, and FAST.

To understand why FAST does not pick up the 11^{th} pair, we plot heat maps of the residuals of selected pairs in Figure 4.6. (x_1, x_2) and (x_2, x_7) are two of the correctly ranked true pairs, (x_1, x_7) is a false pair ranked below the true pairs FAST detects correctly but above the true pair it misses, and (x_8, x_{10}) is the true pair FAST misses and ranks below this false pair. The heat maps show strong interactions are easy to distinguish, but some false interactions such as (x_1, x_7) can have signal as strong as that of weak true interactions such as (x_8, x_{10}). In fact, Sorokina et al. found that x_8 is a weak feature, and do not consider pairs that use x_8 as interactions on $5,000$ samples [60], so we are near the threshold of detectability of (x_8, x_{10}) going from $5,000$ to $10,000$ samples.

Feature Correlation and Spurious Pairs

If features are correlated, spurious interactions may be detected because it is difficult to tell the difference between a true interaction between x_1 and x_2 and the spurious interaction between x_1 and x_3 when x_3 is strongly correlated with x_2; any interaction detection method such as FAST that examines pairs in isolation will have this problem. With GA^{2}M, however, it is fine to include some false positive pairs because GA^{2}M is able to post-filter false positive pairs by looking at the term weights of shaped interactions in the final model.

To demonstrate this, we use the synthetic function in Equation 4.13, but make x_6 correlated to x_1. We generate 2 datasets, one with $\rho(x_1, x_6) = 0.5$ and the other with $\rho(x_1, x_6) = 0.95$, where ρ is the correlation coefficient. We run FAST on residuals after feature shaping. We give the top 20 pairs found by FAST to

GA^{2}M, which then uses gradient boosting to shape those pairwise interactions. Figure 4.7 illustrates how the weights of selected pairwise interactions evolve after each step of gradient boosting. Although the pair (x_2, x_6) can be incorrectly introduced by FAST because of the high correlation between x_1 and x_6, the weight on this false pair decreases quickly as boosting proceeds, indicating that this pair is spurious. This not only allows the model trained on the pairs to remain accurate in the face of spurious pairs, but also reduces the weight (and ranking) given to this shaped term so that intelligibility is not be hurt by the spurious term.

4.5.3 Scalability

Figure 4.5(b) illustrates the running time of different methods on $10,000$ samples from Equation 4.13. Model building time is included. FAST takes about 10 seconds to rank all possible pairs while the two other accurate methods, ANOVA and Grove, are 3-4 orders of magnitude slower. Grove, which is probably the most accurate interaction detection method currently available, takes almost a week to run once on this data. This shows the advantage of FAST; it is very fast with high accuracy. On this problem FAST takes less than 1 second to rank all pairs and the majority of time is devoted to building the additive model.

Figure 4.8 shows the running time of FAST per pair on real datasets. It is clear that on real datasets, FAST is both accurate and efficient.

4.5.4 Design Choices

An alternate to interaction detection that we considered was to build ensembles of trees on residuals after shaping the individual features and then look at

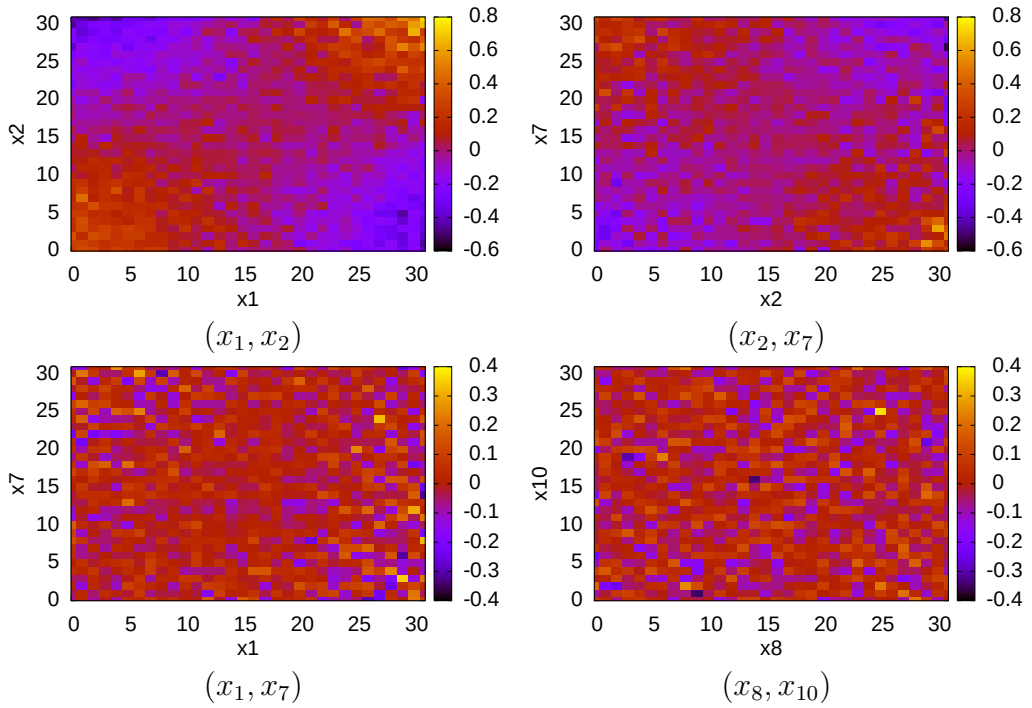
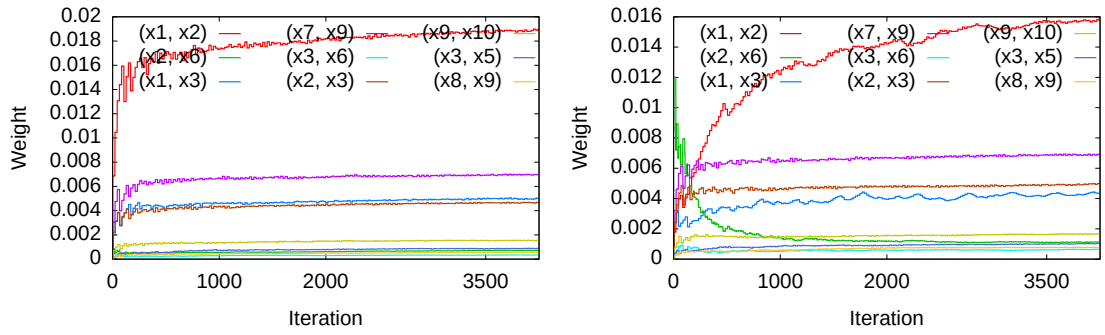


Figure 4.6: True/Spurious heat maps. Features are discretized into 32 bins for visualization.



(a) $\rho(x_1, x_6) = 0.5$ (b) $\rho(x_1, x_6) = 0.95$

Figure 4.7: Weights for pairwise interaction terms in the model.

tree statistics to find combinations of features that co-occur in paths more often than their independent rate warrants. By using 1-step look-ahead at the root we also hoped to partially mitigate the myopia of greedy feature installation to make interactions more likely to be detected. Unfortunately, features with high "co-occurence counts" did not correlate well with true interactions on synthetic test problems, and the best tree-based methods we could devise did not detect

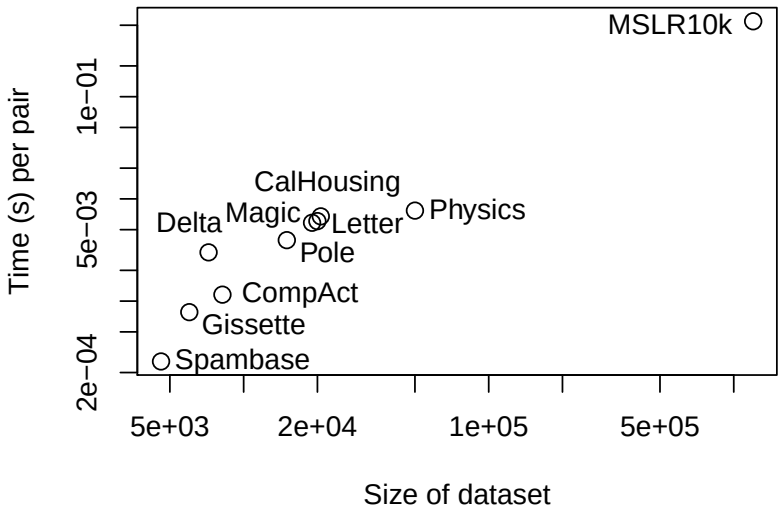


Figure 4.8: Computational cost on real datasets.

interactions as well as FAST, and were considerably more expensive.

4.5.5 Case Study: Learning to Rank

Learning-to-rank is an important research topic in the data mining, machine learning and information retrieval communities. In this section, we train intelligible models with shaped one-dimensional features and pairwise interactions on the "MSLR10k" dataset. A complete description of features can be found in [6]. We show the top 10 most important individual features and their shape functions in first two rows of Figure 4.9. The number above each plot is the weight for the corresponding term in the model. Interestingly, we found BM25 [49], usually considered as a powerful feature for ranking, ranked 70^{th} (BM25_url) in the list after shaping. Other features such as IDF (inverse document frequency) enjoy much higher weight in the learned model.

The last two rows of Figure 4.9 show the 10 most important pairwise interactions and their term strengths. Each of them shows a clear interaction that could not be modeled by additive terms. The non-linear shaping of the individ-

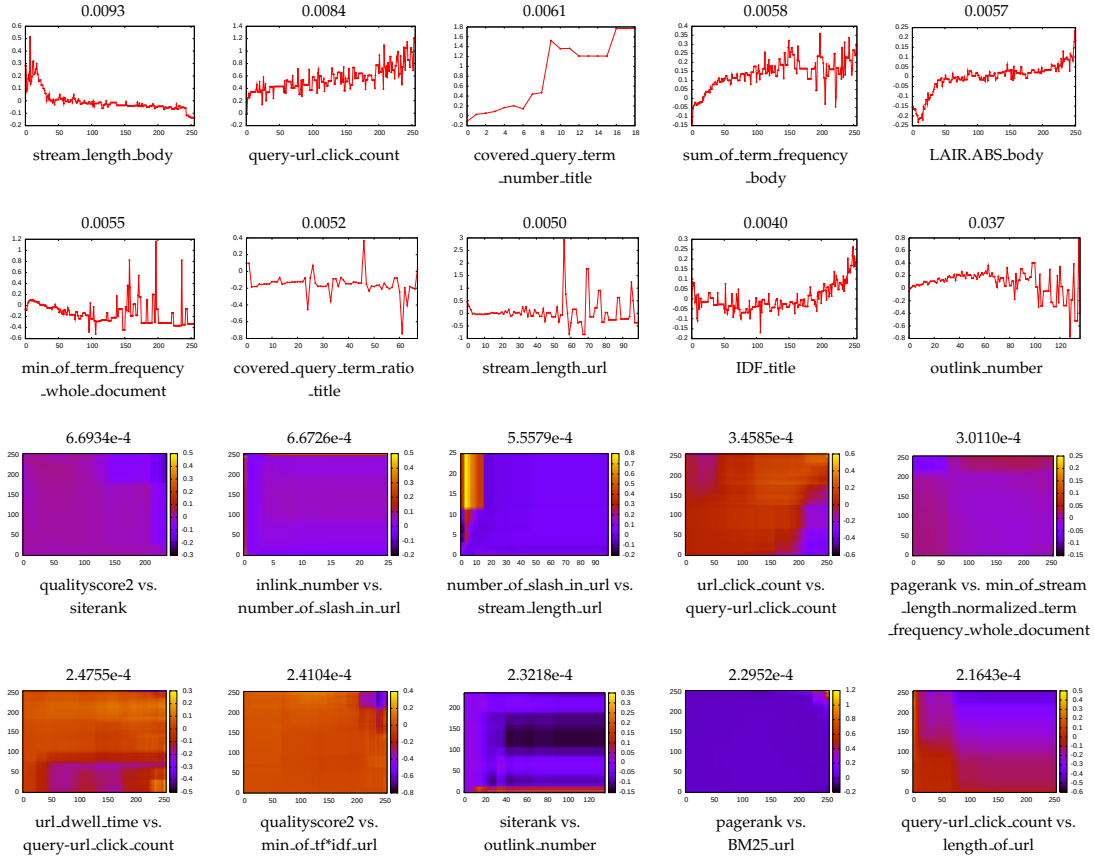


Figure 4.9: Shapes of features and pairwise interactions for the "MSLR10k" dataset with weights. Top two rows show top 10 strongest features. Next two rows show top 10 strongest interactions.

ual features in the top plots and the pairwise interactions in the bottom plots are intelligible to experts and feature engineers, but would be well hidden in full-complexity models.

4.6 Conclusions

We present a framework called GA^{2}M for building intelligible models with pairwise interactions. Adding pairwise interactions to traditional GAMs retains intelligibility, while substantially increasing model accuracy. To scale up pairwise interaction detection, we propose a novel method called FAST that efficiently

measures the strength of all potential pairwise interactions.

CHAPTER 5

GENERALIZED SPARSE PARTIALLY LINEAR ADDITIVE MODELS

A plurality is not to be posited without necessity.

— William of Ockham.

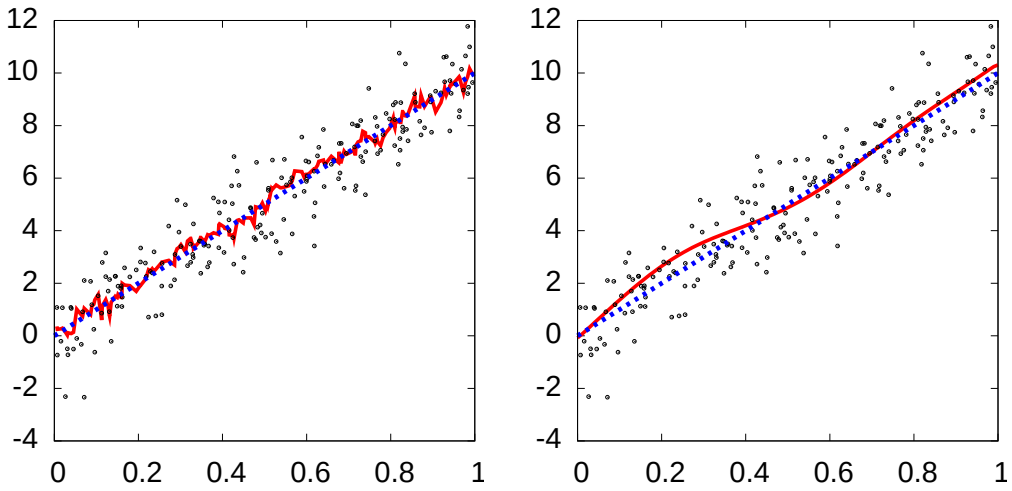
5.1 Introduction

Generalized additive models (GAMs), an extension of generalized linear models (GLMs), model the dependent variable in terms of a sum of univariate functions [32] using the following form,

$$g(E[y]) = \sum f_j(x_j) \tag{5.1}$$

where g is the link function. Since each component f_j is not necessarily linear, GAMs offer greater flexibility than GLMs. However, the low bias associated with GAMs can be outweighed by an increase in variance, in which case GAMs may suffer from overfitting. To prevent overfitting, the parsimonious principle suggests increasing model complexity *only when necessary*. Most previous work, including the prevalent ℓ_1 regularized models [62], seeks parsimony by limiting the number of selected features. While overfitting can be reduced by introducing sparsity constraints across the components (which leads to generalized sparse additive models [58]), little work has been done to intelligently control the complexity within components that are included in the model (e.g., determining whether a component should stay exactly linear or it can be allowed to act nonlinearly).

Explicitly recovering linear components during GAM fitting in addition to variable selection is important for several reasons. First, it significantly reduces the model complexity if most of the true components were linear and therefore



(a) Jagged Tree Fit (b) Nonlinear Spline Fit

Figure 5.1: Fitted $f(x_7)$ for dataset in Example 4. Black dots represent the noisy data points. Estimated function is in red solid line and true (linear) component is in blue dashed line.

reduces overfitting. By carefully modeling the mixture of linear and nonlinear components, the predictive performance is often better than linear-only and nonlinear-only models as it better balances the bias and the variance. Second, prediction on linear components is usually much faster than on their nonlinear counterparts. Third, linear components are more intelligible than nonlinear components [46]; analyzing the slope (a single number) on a linear component is much easier than looking at a near-linear plot in standard GAMs. Intelligibility is also significantly improved when there is only a small subset of the selected components being nonlinear while most selected terms are linear since one would only need to look at a small number of plots.

It is usually hard to recover exactly a linear component using a nonlinear representation in most GAM fitting framework. Non-smooth representation, such as regression trees or ensembles of regression trees [46], is likely to produce a wiggled line for linear terms. Even smooth representations which expand each component into a group of basis functions constructed from a single

covariate [9, 68] will recover linearity only if a linear basis is included. However, most previous work focuses on controlling the *smoothness* of the components rather than the *complexity* (linear vs. nonlinear) of the components [68], and therefore the learned GAM is less intelligible; it may contain lots of nearly linear terms which can *only* be discovered through visualization.[1] When the noise level is large, the situation gets even worse since the component will try to model the noise which forces the model to overfit.

Example 4. *Assume we are given a model* $y = 2\sin(2x_1) + x_2^2 + \exp(-x_3) + x_4 - 3x_5 + 2.5x_6 + 10x_7 + 2x_8 - 7x_9 + 5x_{10} + \epsilon$, *where* $\epsilon \sim \mathcal{N}(0, 1)$. *Figure 5.1 illustrates the fitted component for* x_7 *using ensemble of regression trees and cubic spline. It is obvious that neither of them perfectly recovers the linear term on* x_7*; trees are jagged and splines overfit to the noise. As we will see in Section 5.6.1, explicitly setting some features to be linear better trades off the bias and variance, which leads to a model with better predictive power than GAM.*

GAMs with some of the terms being exactly linear is usually known as generalized partially linear additive models (GPLAMs) [31]. Unfortunately, the state-of-the-art approach of fitting GPLAMs either requires the knowledge of which variables are linear or nonlinear *a priori* [69], or employs expensive statistical tests (such as a bootstrap test) to determine the linearity of a component [22].

In this chapter, we make the following contributions.

- We introduce generalized sparse partially linear additive models (SPLAMs), which automatically select variables and control the complexity (linear vs. nonlinear) of each component during the fitting process.
- We pose the problem as a convex regularization problem, making use of the hierarchical group lasso [70].

[1] Often in this case the weights on the non-linear bases are also not zero.

- We propose two optimization algorithms that use a novel combination of block coordinate gradient descent, block coordinate descent and proximal gradient descent to solve the optimization problem.
- We present a statistical analysis of the theoretical properties of SPLAM.
- We propose a general technique for learning on large data-sparse problems.
- We perform a thorough set of experiments on both synthetic and real datasets that show SPLAMs often outperform GLMs and GAMs in high dimensions when a small subset of features are allowed to be nonlinearly transformed.

The rest of the chapter is organized as follows. We start with preliminaries in Section 5.2. Our method is described in Section 5.3. Theoretical properties are discussed in Section 5.4. Scalability techniques are presented in Section 5.5. Experimental results are reported in Section 5.6, and the chapter concludes in Section 5.7.

5.2 Preliminaries

In this section, we describe preliminaries and review existing work. Let $\mathcal{D} = \{(x_i, y_i)\}_1^N$ denote a dataset of size N, where $x_i = (x_{i1}, ..., x_{ip})$ is the ith feature vector with p features. We use x_j to denote the jth feature in the dataset.

5.2.1 Feature Representation

We represent each component using a group of M basis functions,

$$b(x_j) = [b_1(x_j), ..., b_M(x_j)], \tag{5.2}$$

where $b_1(x_j) = x_j$ is the linear basis function and for $i > 1$, $b_i(x_j)$ represents the nonlinear basis functions. Many choices of basis expansion exist. A cubic spline, for example, is a piecewise cubic polynomial which passes through m control points (called knots) [33]. It can be represented by the truncated power basis,

$$b(x_j) = [x_j, x_j^2, x_j^3, (x_j - x_{j1}^*)_+^3, ..., (x_j - x_{jm}^*)_+^3] \tag{5.3}$$

where $(\cdot)_+$ represents the non-negative part; however, other bases, such as B-splines, are often used because they have better computational properties.

For simplicity, we ignore the intercept and expand the input space to $b(x_i) = [b(x_{i1}), ..., b(x_{ip})]$. The dimension of the augmented space is denoted as $p_+ = pM$ and the design matrix is denoted as X, where $X_i = b(x_i)$ is the ith row in X. Let $\mathcal{G}_j$ be the group of indices for all variables on block j ($|\mathcal{G}_j| = M$) and denote X_j as the columns indexed by $\mathcal{G}_j$. Similarly, let $\mathcal{G}_{j,-1}$ be the group of indices for nonlinear variables on block j ($|\mathcal{G}_{j,-1}| = M - 1$).

For regression problems, it is also popular to consider orthonormal basis expansions, i.e., $X_j^T X_j = I_M$. In Section 5.3.1, we will use this feature representation to exploit a faster optimization procedure.

5.2.2 Hierarchical Sparsity Regularization

Let $\beta_j \in \mathbb{R}^M$ denote the weight vector for group $\mathcal{G}_j$ and $\beta = [\beta_1^T, ..., \beta_p^T]^T \in \mathbb{R}^{p_+}$. We use $\beta_{j,1}$ to denote the weight on the linear basis $b_1(x_j)$, and $\beta_{j,-1}$ for the weight vector on the nonlinear bases $[b_2(x_j), ..., b_M(x_j)]$. Thus, the weight vector $\beta_j \neq 0$ is equivalent to x_j being selected in the model. When $\beta_{j,1} \neq 0$ and $\beta_{j,-1} = 0$, the component stays exactly linear. Such hierarchical sparsity constraints naturally lead to a hierarchical group lasso-like formulation [70], where the group on nonlinear bases is nested in the group of all the bases on x_j.

Given a convex smooth loss function $f(\beta)$, we formulate the optimization function as follows,

$$\min_{\beta} F(\beta) \stackrel{\text{def}}{=} f(\beta) + h(\beta) \tag{5.4}$$

where $h(\beta) = \lambda\Omega(\beta) = \lambda \sum_j p(\beta_j)$ with $\lambda > 0$ and $p(\beta_j)$ is the following convex but non-smooth penalization term ($\alpha \in [0, 1]$),

$$p(\beta_j) = \alpha\|\beta_j\|_2 + (1 - \alpha)\|\beta_{j,-1}\|_2. \tag{5.5}$$

Note that ℓ_2 norm is a natural extension to the ℓ_1 norm on the group-wise case to enforce sparsity. When $\alpha = 1$, this regularization term does not penalize the weights on nonlinear bases and therefore in this case we only select features and each selected features will be nonlinear. When $\alpha \to 0$, this regularization term penalizes a lot on the weights for nonlinear bases, and in this case we will get a sparse (almost) linear model.

As shown in [70], this penalization enforces hierarchical sparsity. We call models of this form *generalized sparse partially linear additive models* (SPLAMs).

For regression problems, $f(\beta) = \frac{1}{2N} \sum_i (y_i - X_i\beta)^2$ is the quadratic loss and for binary classification problems, $f(\beta) = \frac{1}{N} \sum_i \log(1+\exp(-y_i(X_i\beta)))$ is logistic loss, where $y_i \in \{-1, 1\}$.

Note that our formulation is different from the standard tree-structured group lasso in that we do not require the union of all subgroups to be the whole group [39, 44]. In fact, the only subgroup in our formulation is the group on nonlinear bases. In addition, the nature of our problem is inherently hierarchical, rather than some user-defined hierarchy [36, 39].

5.2.3 Proximal Gradient Method

A standard approach to solving convex problems with hierarchical sparsity regularization is via the proximal gradient method [11, 36]. In this section, we briefly review this method, which will serve as a building block for our algorithms.

At each iteration k, f is linearized around the current estimate β^k, and the next estimate β^{k+1} is updated as the solution of the following proximal problem,

$$\beta^{k+1} = \arg\min_{z \in \mathbb{R}^{p+}} \{f(\beta^k) + \langle z - \beta^k, \nabla f(\beta^k) \rangle + \frac{1}{2t^k} \|z - \beta^k\|_2^2 + h(z)\} \tag{5.6}$$

where $t^k > 0$ is a suitable step size. Ignoring the constant terms, this problem can be rewritten as,

$$\beta^{k+1} = P_{t^k}(\beta^k), \tag{5.7}$$

where

$$P_t(\beta) = \arg\min_{z \in \mathbb{R}^{p+}} \{\frac{1}{2t} \|z - (\beta - t\nabla f(\beta))\|_2^2 + h(z)\} \tag{5.8}$$

is the proximal operator evaluated at $\beta - t\nabla f(\beta)$. The proximal problem with a hierarchical regularizer can be solved very efficiently using a primal-dual approach [36].

Setting the step size to $1/L$ is guaranteed to converge [11], where L is the Lipschitz constant of ∇f. For example, for quadratic loss $f(\beta) = \frac{1}{2N} \sum_i (y_i - X_i\beta)^2$, $L = \frac{1}{N} \|X^T X\|_2$.

However, computing the Lipschitz constant is not always easy and using a fixed step size can be too conservative. We could use backtracking line search [11] to pick as large a step as possible while still guaranteeing the convergence: start with a large step size t^0 and some $\eta \in (0, 1)$, find the smallest i_k

such that with $\bar{t} = \eta^{i_k} t^0$, the following inequality holds,

$$f(\beta'^k) \leq f(\beta^k) + \langle \beta'^k - \beta^k, \nabla f(\beta^k) \rangle + \frac{1}{2\bar{t}} \|\beta'^k - \beta^k\|_2^2, \tag{5.9}$$

where $\beta'^k = P_{\bar{t}}(\beta^k)$.

This proximal gradient method is also known as iteratively shrinkage thresholding algorithm (ISTA), and enjoys a convergence rate of $O(1/T)$, where T is the number of iterations. The convergence rate can be improved to $O(1/T^2)$ by adding a momentum using Nesterov's method [11, 52]. This method is usually called fast ISTA (FISTA).

5.3 Our Approach

5.3.1 Optimization

In this chapter, we consider different approaches using block coordinate gradient descent (BCGD) and block coordinate descent (BCD) to fit SPLAM. We will first describe the general method of solving SPLAM using BCGD in Section 5.3.1. For regression problems, we exploit the property of the quadratic loss and propose a more efficient BCD method in Section 5.3.1.

Block Coordinate Gradient Descent

We first propose a BCGD method to solve SPLAM. For each block j, we form the proximal operator and use its solution as our new estimate of β_j for block j. A natural extension to the proximal gradient method in this BCGD framework is to allow different step sizes for each block j. Thus, the subproblem becomes

$$\beta_j^{k+1} = P_{t_j}^j(\beta^k) \tag{5.10}$$

where t_j is the step size for block j and

$$P_t^j(\beta) = \arg\min_{z \in \mathbb{R}^M} \{\frac{1}{2t}\|z - (\beta_j - t\nabla_j f(\beta))\|_2^2 + \lambda p(z)\}, \tag{5.11}$$

and $\nabla_j f(\cdot) \in \mathbb{R}^M$ is the gradient vector on block j. Note that with the BCGD framework, computing the Lipschitz constant L_j for $\nabla_j f$ is no longer expensive; $X_j^T X_j$ is just an M-by-M matrix, where M is typically very small. Thus, we also compute the minimum step size $1/L_j$ to avoid the step size t_j going below this value.

This proximal problem can be solved very efficiently using a primal-dual approach [36]. Let $g_j = \beta_j^k - t_j \nabla_j f(\beta^k)$ and consider the dual problem,

$$\min_{\gamma_1, \gamma_2} \quad \frac{1}{2}\|g_j - \gamma_1 - [0, \gamma_2^T]^T\|_2^2 \tag{5.12}$$

$$s.t. \quad \|\gamma_1\|_2 \leq t_j \lambda \alpha \tag{5.13}$$

$$\|\gamma_2\|_2 \leq t_j \lambda (1 - \alpha) \tag{5.14}$$

where $\gamma_1 \in \mathbb{R}^M$ and $\gamma_2 \in \mathbb{R}^{M-1}$.

As shown in [36], this dual problem can be solved in *one pass* of block coordinate descent as follows,

$$\gamma_2 = \Pi_{t_j \lambda (1-\alpha)}(g_{j,-1}) \tag{5.15}$$

$$\gamma_1 = \Pi_{t_j \lambda \alpha}(g_j - [0, \gamma_2^T]^T) \tag{5.16}$$

where $\Pi_r(u)$ projects the vector u onto the ball of radius r. The solution to the primal is $z = g_j - \gamma_1 - [0, \gamma_2^T]^T$.

We perform a backtracking line search to ensure the following inequality holds,

$$f(\tilde{\beta}) \leq f(\hat{\beta}) + \langle \beta_j' - \beta_j^k, \nabla_j f(\hat{\beta}) \rangle + \frac{1}{2t_j}\|\beta_j' - \beta_j^k\|_2^2, \tag{5.17}$$

Algorithm 11 SPLAM via BCGD

1: $t_j = t_j^0$, for $j = 1, ..., p$
2: $k \leftarrow 0$
3: $\beta^0 \leftarrow 0$
4: **while** not converge **do**
5: **for** $j = 1$ to p **do**
6: **while true do**
7: $g_j \leftarrow \beta_j^k - t_j \nabla_j f(\beta^k)$
8: $\gamma_2 \leftarrow \Pi_{t_j \lambda(1-\alpha)}(g_{j,-1})$
9: $\gamma_1 \leftarrow \Pi_{t_j \lambda \alpha}(g_j - [0, \gamma_2^T]^T)$
10: $\beta_j^{k+1} \leftarrow g_j - \gamma_1 - [0, \gamma_2^T]^T$
11: **if** Inequality 5.17 holds **then**
12: **break**
13: **else**
14: $t_j \leftarrow \min(\eta t_j, 1/L_j)$
15: $k \leftarrow k + 1$

where

$$\beta_j' = P_{t_j}^j(\beta^k) \tag{5.18}$$

$$\hat{\beta} = [\beta_1^{k+1^T}, ..., \beta_{j-1}^{k+1^T}, \beta_j^{k^T}, ..., \beta_p^{k^T}]^T \tag{5.19}$$

$$\tilde{\beta} = [\beta_1^{k+1^T}, ..., \beta_{j-1}^{k+1^T}, \beta_j'^T, ..., \beta_p^{k^T}]^T. \tag{5.20}$$

Algorithm 11 summarizes our block coordinate gradient descent method. We cycle through all blocks (Line 5), solve the proximal operator for that block (Line 7-10) and check if the step size is proper using a backtracking line search (Line 11-14).

Block Coordinate Descent

Although Algorithm 11 is applicable to any differentiable loss function f (including quadratic loss), we propose a more efficient BCD approach to solve regression problems by exploiting the property of quadratic loss.

Consider an orthonormal basis expansion where each block j in the design matrix X, in this case denoted as Q_j, is orthonormal, i.e., $Q_j^T Q_j = I_M$. In block

coordinate descent, for quadratic loss each subproblem on block j can be formalized using matrix form as follows,

$$\min_{\beta_j} \frac{1}{2N}\|r_j - Q_j\beta_j\|_2^2 + \lambda\alpha\|\beta_j\|_2 + \lambda(1-\alpha)\|\beta_{j,-1}\|_2 \tag{5.21}$$

where $r_j = y - \sum_{k\neq j} Q_k\beta_k$ is the partial residual.

Using the orthonormality of Q_j, we observe that this problem has the same minimizer as.

$$\min_{\beta_j} \frac{1}{2N}\|Q_j^T r_j - \beta_j\|_2^2 + \lambda\alpha\|\beta_j\|_2 + \lambda(1-\alpha)\|\beta_{j,-1}\|_2 \tag{5.22}$$

Notice that this is the proximal operator applied on $Q_j^T r_j$ (instead of $\beta_j - t\nabla_j f(\beta)$), which can be similarly solved in one pass in the dual form. Thus, we completely eliminate the need to use the step size and the back tracking line search in BCGD, which makes optimization much more efficient.

We perform a QR decomposition for each block j using Gram-Schmidt process, in order to preserve the linear basis in the first column of each block, i.e., $X_j = Q_j R_j$, for any X_j. Algorithm 12 summarizes our BCD algorithm. With suitable choices of $b(x_j)$ (such as the cubic spline bases described in Section 5.2.1), such QR decomposition usually works well in practice and R_j is usually invertible.

Note that we are solving a different optimization problem and this solution is not the optimal solution to the original problem. Nevertheless, our choice of feature representation is fairly arbitrary. From the basis expansion's perspective, all that matters is the space spanned by X_j. Our penalty, however, is sensitive to the basis that we are working with so it is important to preserve the linear basis in the first column of each block.

Algorithm 12 SPLAM via BCD

$\beta^0 \leftarrow 0$
while not converge **do**
  for $j = 1$ to p **do**
    $r_j \leftarrow y - \sum_{k \neq j} Q_k \beta_k$
    $g_j \leftarrow Q_j^T r_j$
    $\gamma_2 \leftarrow \Pi_{t_j \lambda(1-\alpha)}(g_{j,-1})$
    $\gamma_1 \leftarrow \Pi_{t_j \lambda \alpha}(g_j - [0, \gamma_2^T]^T)$
    $\beta_j \leftarrow g_j - \gamma_1 - [0, \gamma_2^T]^T$

5.3.2 Practical Issues

Active Set Strategy

We employ the widely used active set strategy [28, 40, 50]. After a complete cycle through all the variables, we iterate only on the active set till convergence. If another complete cycle does not change the active set, we are done, otherwise the process is repeated.

Refitting Strategy

We also employ a refitting strategy. Given a pair of (λ, α), once the optimization converges, we keep the learned structure (linear and nonlinear components), and refit the model without regularization parameters. In this case, we simply use λ and α to search the model structure. Once we know the model structure, the problem reduces to linear regression or logistic regression on a different design matrix. Our experiment shows this refitting strategy works well in practice and usually produces better models when there is enough data for reliable estimates.

Algorithm 13 Finding λ_{max}

1: $\lambda_h \leftarrow \max_j \frac{\|\nabla_j f(0)\|_2}{\alpha}$
2: $\lambda_l \leftarrow 0$
3: **while** $\lambda_h - \lambda_l \geq \epsilon$ **do**
4: $\quad \lambda \leftarrow \frac{\lambda_h + \lambda_l}{2}$
5: $\quad$ **if** $\forall j, P_t^j(0) = 0$ **then**
6: $\quad\quad \lambda_h \leftarrow \lambda$
7: $\quad$ **else**
8: $\quad\quad \lambda_l \leftarrow \lambda$
9: $\lambda_{max} = \lambda_h$

Regularization Path

Similar to `glmnet` [28], the optimization of SPLAM also uses two parameters, λ and α, which usually involves a grid search on values of (λ, α) pairs. Our strategy is to generate a complete regularization path with some α fixed, and therefore we need to find the smallest value λ_{max} for which $\beta_j = 0$ for $j = 1, ..., p$. Starting from λ_{max}, we decrease λ exponentially until we activate all components. Once we have a set of model structures, we refit all the models and choose the best model on a held-out validation set.

The key question is how to find λ_{max}. Notice that for all $\lambda \geq \lambda_{init} = \max_j \frac{\|\nabla_j f(0)\|_2}{\alpha}$, zero point will be the solution to our optimization problem. Therefore, we perform a binary search to find λ_{max}. As described in Algorithm 13, we start with λ_{init} (Line 1) and effectively shrink the interval $[\lambda_l, \lambda_h]$ (Line 3 - 8) to locate λ_{max}.

5.4 Theoretical Properties

5.4.1 Convergence

While it is apparent that Algorithm 12 is the standard block coordinate descent algorithm with guaranteed convergence, it is not obvious that Algorithm 11 also converges since we are running proximal operator at the block level. In this section, we show that Algorithm 11 fits the general BCGD framework [63] and therefore the global convergence is guaranteed. A similar convergence result for group lasso has also been shown [55]. We first briefly review the general BCGD algorithm.

At each iteration k, for block j, choose a symmetric matrix H^k, and compute the search direction.

$$d^k = \arg\min_d \{\nabla f(\beta^k)^T d + \frac{1}{2} d^T H^k d + h(\beta^k + d)\} \tag{5.23}$$

where $\forall i \notin \mathcal{G}_j, d_i = 0$. Then a step size $\alpha^k > 0$ is chosen so that the following Armijo rule is satisfied,

$$F(\beta^k + \alpha^k d^k) \leq F(\beta^k) + \alpha^k \sigma \Delta^k \tag{5.24}$$

where $0 < \sigma < 1, 0 \leq \gamma < 1$, and

$$\Delta^k \stackrel{\text{def}}{=} \nabla f(\beta^k)^T d^k + \gamma {d^k}^T H^k d^d + h(\beta^k + d^k) - h(\beta^k), \tag{5.25}$$

Once the step size α^k is determined, update $\beta^{k+1} = \beta^k + \alpha^k d^k$.

Theorem 2 in [63] guarantees the global convergence when $\overline{\theta} I \succeq H^k \succeq \underline{\theta} I$, $0 < \underline{\theta} \leq \overline{\theta}$.

Theorem 1. *Algorithm 11 fits the general BCGD framework [63] . The global convergence is guaranteed and Algorithm 11 converges Q-linearly.*

Proof. First, for block j, setting $H^k = \frac{1}{t_j} I$, Equation (5.23) is equivalent to our proximal operator for block j after ignoring constants. Next, notice that when $\alpha^k = 1, \sigma = 1$, and $\gamma = \frac{1}{2}$, the Armijo rule becomes our backtracking line search step in Inequality (5.17). That is, the effort of choosing step size is shifted to finding H^k. Besides, Lemma 1 in [63] suggests $\nabla f(\beta^k)^T d^k + {d^k}^T H^k d^k + h(\beta^k + d^k) - h(\beta^k) \leq 0$. Since $H^k \succ 0$, with $\gamma = \frac{1}{2}$, we can easily see $\Delta^k \leq 0$ whenever $d^k \neq 0$, which means if the Armijo rule holds for $\sigma = 1$, it must also hold for $\sigma < 1$. Finally, we show that $\overline{\theta} I \succeq H^k \succeq \underline{\theta} I$. Assume the initial step size is t_j^0, this is true when $\overline{\theta} = \max\{L_j, 1/t_j^0\}$ and $\underline{\theta} = \min\{L_j, 1/t_j^0\}$. Thus, according to Theorem 2 in [63], Algorithm 11 converges Q-linearly. □

5.4.2 An Oracle Inequality

In this section, we seek a deeper understanding of the regimes in which SPLAM works well, and to this end we prove an oracle inequality, giving an upper bound on its prediction error in the regression setting.

Suppose $y = \sum_{j=1}^{p} X_j \beta_j^0 + \epsilon$, where X_j denotes the $N \times M$ matrix for the jth feature, $\beta_j^0 \in \mathbb{R}^M$ is the true vector of coefficients, and $\epsilon \sim N(0, \sigma^2 I_N)$ is a random vector of noise. We will assume in this section that we are using an orthogonal basis for each feature j, i.e., that $\frac{1}{N} X_j^T X_j = I_M$.[2] We describe the sparsity of the vector $\beta^0 \in \mathbb{R}^{p+}$ in two senses: first, whether a feature is at all relevant, $S_0 = \{j : \beta_j^0 \neq 0\}$, and second, whether the feature is nonlinear, $N_0 = \{j : \beta_{j,-1}^0 \neq 0\}$. We may also define the set of linear features, $L_0 = S_0 \setminus N_0$. Given some set S, we use β_S to denote the sub-vector indexed by S. For simplicity, we define $\lambda_1 = \lambda\alpha$ and $\lambda_2 = \lambda(1-\alpha)$ in this section. Recall that our penalty is

[2]Note this is different from the orthonormal basis in Section 5.3.1, but this is just a rescaling of each block which makes our proof easier.

$\Omega(\beta) = \alpha \sum_{j=1}^{p} \|\beta_j\|_2 + (1-\alpha) \sum_{j=1}^{p} \|\beta_{j,-1}\|_2$. Let

$$\hat{\beta} \in \arg\min_{\beta} \frac{1}{2N} \|y - \sum_j X_j \beta_j\|_2^2 + \lambda \Omega(\beta). \tag{5.26}$$

Theorem 2. *If we take* $\lambda \geq 2(1+2\sqrt{6})\sigma\sqrt{\log p/N}$ *and* $\alpha = (1+\sqrt{6})/(1+2\sqrt{6})$, *then*

$$\frac{1}{N}\|X\hat{\beta} - X\beta^0\|_2^2 \leq 3\lambda \left[\alpha \sum_{j \in L^0} |\beta_{j1}^0| + \sum_{j \in N^0} \|\beta_j\|_2 \right] \tag{5.27}$$

holds with probability at least $1 - 4/p$ *as long as* $\log p > M/8$.

Proof. By definition of $\hat{\beta}$,

$$\frac{1}{2N}\|y - X\hat{\beta}\|_2^2 + \lambda\Omega(\hat{\beta}) \leq \frac{1}{2N}\|y - X\beta^0\|_2^2 + \lambda\Omega(\beta^0),$$

which after some algebra (writing $\hat{\Delta} = \hat{\beta} - \beta^0$) leads to

$$\frac{1}{2N}\|X\hat{\Delta}\|_2^2 + \lambda\Omega(\hat{\beta}) \leq \frac{1}{N}\epsilon^T X\hat{\Delta} + \lambda\Omega(\beta^0) \tag{5.28}$$

Define the empirical process as,

$$V_N(\hat{\Delta}) = \frac{1}{N}\epsilon^T X\hat{\Delta} = \frac{1}{\sqrt{N}} \sum_{j=1}^{p} V_j^T \hat{\Delta}_j \tag{5.29}$$

where $V_j = \frac{1}{\sqrt{N}} X_j^T \epsilon \in \mathbb{R}^M$.

Now we bound the empirical process. First we notice that,

$$\begin{aligned} |V_j^T\hat{\Delta}_j| \leq & \frac{1}{2}\left[|V_j^T\hat{\Delta}_j| + |V_{j1}\hat{\Delta}_{j1}| + |V_{j,-1}^T\hat{\Delta}_{j,-1}| \right] & (5.30) \\ \leq & \frac{1}{2}\left[\|V_j\|_2\|\hat{\Delta}_j\|_2 + |V_{j1}||\hat{\Delta}_{j1}| + \|V_{j,-1}\|_2\|\hat{\Delta}_{j,-1}\|_2 \right] & (5.31) \end{aligned}$$

Thus $|V_N(\hat{\Delta})|$ can be bounded as follows.

$$|V_N(\hat{\Delta})| \leq \frac{1}{\sqrt{N}} \sum_{j=1}^{p} |V_j^T \hat{\Delta}_j| \tag{5.32}$$

$$\leq \frac{1}{2\sqrt{N}} \left(\max_j \|V_j\|_2 \|\hat{\Delta}\|_{2,1} + \max_j |V_{j1}| \sum_j |\hat{\Delta}_{j1}| + \max_j \|V_{j,-1}\|_2 \|\hat{\Delta}_{\cdot,-1}\|_{2,1} \right) \tag{5.33}$$

$$\leq \frac{1}{2\sqrt{N}} \left[(\max_j \|V_j\|_2 + \max_j |V_{j1}|) \|\hat{\Delta}\|_{2,1} + \max_j \|V_{j,-1}\|_2 \|\hat{\Delta}_{\cdot,-1}\|_{2,1} \right] \tag{5.34}$$

Observing that $V_j \sim N(0, \sigma^2 I_M)$, we have $\|V_j\|_2^2 \sim \sigma^2 \chi_M^2$. Thus, by Lemma 6.2 and 8.1 of [18], we have

$$P\left(\frac{\max_j |V_{j1}|}{2\sqrt{N}} > \nu_1 \right) \leq 2e^{-x} \tag{5.35}$$

$$P\left(\frac{\max_j \|V_j\|_2}{2\sqrt{N}} > \nu_2 \right) \leq e^{-x} \tag{5.36}$$

$$\tag{5.37}$$

where,

$$\nu_1^2 = \frac{\sigma^2}{2N}(x + \log p) \tag{5.38}$$

$$\nu_2^2 = \frac{\sigma^2}{4N} \left[M + \sqrt{4M(x + \log p)} + 4(x + \log p) \right] \tag{5.39}$$

Thus, we have

$$P(\frac{\max_j \|V_j\|_2 + \max_j |V_{j1}|}{2\sqrt{N}} > \nu_1 + \nu_2) \leq 3e^{-x} \tag{5.40}$$

$$P(\frac{\max_j \|V_{j,-1}\|_2}{2\sqrt{N}} > \nu_2) \leq e^{-x} \tag{5.41}$$

Therefore (with union bound),

$$P\left(|V_N(\hat{\Delta})| \leq \left[(\nu_1 + \nu_2) \|\hat{\Delta}\|_{2,1} + \nu_2 \|\hat{\Delta}_{\cdot,-1}\|_{2,1} \right] \right) \geq 1 - (e^{-x} + 3e^{-x}) \tag{5.42}$$

$$= 1 - 4e^{-x} \tag{5.43}$$

Thus, by (5.28) we have with probability at least $1 - 4e^{-x}$ that

$$\frac{1}{2N}\|X\hat{\Delta}\|_2^2 + \lambda\Omega(\hat{\beta}) \leq (\nu_1 + \nu_2)\|\hat{\Delta}\|_{2,1} + \nu_2\|\hat{\Delta}_{\cdot,-1}\|_{2,1} + \lambda\Omega(\beta^0) \tag{5.44}$$

Recall that $\lambda_1 = \lambda\alpha$ and $\lambda_2 = \lambda(1-\alpha)$, we can take $\lambda_1 = 2(\nu_1 + \nu_2)$ and $\lambda_2 = 2\nu_2$. Thus, (5.44) implies

$$\frac{1}{2N}\|X\hat{\Delta}\|_2^2 \leq (\lambda/2)\Omega(\hat{\Delta}) - \lambda\Omega(\hat{\beta}) + \lambda\Omega(\beta^0) \tag{5.45}$$

$$\leq (\lambda/2)[\Omega(\hat{\Delta}) - \Omega(\hat{\beta})] + \lambda\Omega(\beta^0) \tag{5.46}$$

which, by the triangle inequality, gives

$$\frac{1}{N}\|X\hat{\Delta}\|_2^2 \leq \lambda[\Omega(\hat{\beta} - \hat{\Delta})] + 2\lambda\Omega(\beta^0) = 3\lambda\Omega(\beta^0). \tag{5.47}$$

By choosing $x = \log p$, we can ensure our inequality holds with probability at least $1 - 4/p$. This means,

$$\nu_1^2 = \frac{\sigma^2}{N}\log p \tag{5.48}$$

$$\nu_2^2 = \frac{\sigma^2}{4N}\left[M + \sqrt{8M\log p} + 8\log p\right]. \tag{5.49}$$

Define $\tilde{\nu}_1^2 \stackrel{\text{def}}{=} \frac{\sigma^2}{N}\log p$ and notice that $\nu_2^2 \leq 6\sigma^2\log p/N \stackrel{\text{def}}{=} \tilde{\nu}_2^2$ if $\log p \geq M/8$. Now, as long as $\log p \geq M/8$, we can take $\lambda \geq 2(\tilde{\nu}_1 + 2\tilde{\nu}_2) = 2(1+2\sqrt{6})\sigma\sqrt{\log p/N}$ and

$$\alpha = \frac{\tilde{\nu}_1 + \tilde{\nu}_2}{\tilde{\nu}_1 + 2\tilde{\nu}_2} = \frac{1+\sqrt{6}}{1+2\sqrt{6}}, \tag{5.50}$$

with probability at least $1 - 4/p$, we have

$$\frac{1}{N}\|X\hat{\Delta}\|_2^2 \leq \lambda[\Omega(\hat{\beta} - \hat{\Delta})] + 2\lambda\Omega(\beta^0) = 3\lambda\Omega(\beta^0). \tag{5.51}$$

Finally, observe that

$$\Omega(\beta^0) = \alpha \sum_{j \in S^0} \|\beta_j^0\|_2 + (1-\alpha) \sum_{j \in N^0} \|\beta_{j,-1}^0\|_2 \tag{5.52}$$

$$\leq \alpha \sum_{j \in L^0} \|\beta_j^0\|_2 + \sum_{j \in N^0} \|\beta_j^0\|_2 \tag{5.53}$$

$$= \alpha \sum_{j \in L^0} |\beta_{j1}^0| + \sum_{j \in N^0} \|\beta_j^0\|_2 \tag{5.54}$$

□

Corollary 1. *Suppose* $|\beta_k^0| \leq B < \infty$ *for all* $k \in \{1, ..., pM\}$. *Then, for* λ *and* α *as in the previous theorem,*

$$\frac{1}{N}\|X\hat{\beta} - X\beta^0\|^2 \leq 36\sigma B\sqrt{\frac{\log p}{N}}\left[\alpha|L_0| + \sqrt{M}|N_0|\right] \tag{5.55}$$

holds with probability at least $1 - 4/p$ *as long as* $\log p > M/8$.

Proof. We use that $3\lambda \leq 36\sigma\sqrt{\log p/N}$. □

Remarks:

1. The corollary tells us that $\frac{1}{N}\|X\hat{\beta} - X\beta^0\|^2 \to 0$ in probability as $N \to \infty$ even if we let p grow like e^{N^γ} with $\gamma < 1$. It also shows that our error grows linearly both in the number of linear and nonlinear features in the true model.
2. The oracle inequality given above can be greatly improved under stronger assumptions. That said, an appealing feature of the above theorem is that there are no conditions on X other than each $\frac{1}{\sqrt{n}}X_j$ being orthogonal. In fact, in the context of the Lasso such so-called "slow rates" have been shown to be much better than the "fast rates" in certain regimes [65].

Next we show "fast rates" under stronger assumption. We first present a compatibility condition.

Assumption 1 (Compatibility condition)**.**

$$\forall \Delta \in \{\Delta | \alpha \|\Delta_{S_0^c}\|_{2,1} + (1-\alpha)\|\Delta_{N_0^c,-1}\|_{2,1} \leq 3(\alpha \|\Delta_{S_0}\|_{2,1} + (1-\alpha)\|\Delta_{N_0,-1}\|_{2,1})\} \tag{5.56}$$

$\exists \phi_{S_0}, \phi_{N_0} > 0$,

$$\frac{1}{N}|\|X\Delta\|_2^2 \geq \frac{\phi_{S_0}^2(\|\Delta_{S_0}\|_{2,1})^2}{s_0} \tag{5.57}$$

$$\frac{1}{N}|\|X\Delta\|_2^2 \geq \frac{\phi_{N_0}^2(\|\Delta_{N_0,-1}\|_{2,1})^2}{n_0} \tag{5.58}$$

where $s_0 = |S_0|$ *and* $n_0 = |N_0|$.

This condition is a type of restricted eigenvalue condition. It suggests that the minimum eigenvalue is bounded below from some value strictly larger than zero in some restricted area. If the design matrix X is formed by independently sampling each row $X_i \sim N(0, \Sigma)$, referred to as the Σ-Gaussian ensemble, then with high probability that $\frac{1}{N}|\|X\Delta\|_2^2 \geq c$, where $c > 0$ [57]. However, our design matrix is not Σ-Gaussian ensemble and therefore the compatibility condition is a strong assumption.

Theorem 3. *With the compatibility condition, if we take* $\lambda \geq 2(1+2\sqrt{6})\sigma\sqrt{\log p/N}$ *and* $\alpha = (1+\sqrt{6})/(1+2\sqrt{6})$*, then*

$$\frac{1}{N}\|X\hat{\beta} - X\beta^0\|_2^2 \leq 32\lambda^2\left(\frac{\alpha^2 s_0}{\phi_{S_0}^2} + \frac{(1-\alpha)^2 n_0}{\phi_{N_0}^2}\right) \tag{5.59}$$

$$\Omega(\hat{\beta} - \beta^0) \leq 16\lambda\left(\frac{\alpha^2 s_0}{\phi_{S_0}^2} + \frac{(1-\alpha)^2 n_0}{\phi_{N_0}^2}\right) \tag{5.60}$$

hold with probability at least $1 - 4/p$ *as long as* $\log p > M/8$.

Proof. Notice that $\|\hat{\beta}_{L_0,-1}\|_2 = \|\hat{\Delta}_{L_0,-1}\|_{2,1}$ and $\Omega(\hat{\beta}_{S_0^c}) = \Omega(\hat{\Delta}_{S_0^c})$, (5.44) is equivalent to,

$$\frac{1}{2N}\|X\hat{\Delta}\|_2^2 + \lambda_1\|\hat{\beta}_{L_0}\|_{2,1} + \lambda_2\|\hat{\Delta}_{L_0,-1}\|_{2,1} + \lambda\Omega(\hat{\beta}_{N_0}) + \lambda\Omega(\hat{\Delta}_{S_0^c}) \tag{5.61}$$

$$\leq (\nu_1+\nu_2)\|\hat{\Delta}\|_{2,1} + \nu_2\|\hat{\Delta}_{\cdot,-1}\|_{2,1} + \lambda\Omega(\beta^0) \tag{5.62}$$

Thus, we have

$$\frac{1}{2N}\|X\hat{\Delta}\|_2^2 \leq (\nu_1+\nu_2-\lambda_1)\|\hat{\Delta}_{S_0^c}\|_{2,1} + (\nu_1+\nu_2)\|\hat{\Delta}_{S_0}\|_{2,1} \tag{5.63}$$

$$+ (\nu_2-\lambda_2)\|\hat{\Delta}_{N_0^c,-1}\|_{2,1} + \nu_2\|\hat{\Delta}_{N_0,-1}\|_{2,1} \tag{5.64}$$

$$+ \lambda_1\left[\|\beta^0\|_{2,1} - \|\hat{\beta}_{L_0}\|_{2,1} - \|\hat{\beta}_{N_0}\|_{2,1}\right] \tag{5.65}$$

$$+ \lambda_2\left[\|\beta^0_{\cdot,-1}\|_{2,1} - \|\hat{\beta}_{N_0,-1}\|_{2,1}\right] \tag{5.66}$$

$$\tag{5.67}$$

Notice that by the triangle inequality, we have

$$\|\beta^0\|_{2,1} - \|\hat{\beta}_{L_0}\|_{2,1} - \|\hat{\beta}_{N_0}\|_{2,1} = \|\beta^0_{S_0}\|_{2,1} - \|\hat{\beta}^0_{S_0}\|_{2,1} \tag{5.68}$$

$$\leq \|\hat{\beta}_{S_0} - \beta^0_{S_0}\|_{2,1} \tag{5.69}$$

$$= \|\hat{\Delta}_{S_0}\|_{2,1} \tag{5.70}$$

$$\|\beta^0_{\cdot,-1}\|_{2,1} - \|\hat{\beta}_{N_0,-1}\|_{2,1} = \|\beta^0_{N_0,-1}\|_{2,1} - \|\hat{\beta}_{N_0,-1}\|_{2,1} \tag{5.71}$$

$$\leq \|\hat{\beta}_{N_0,-1} - \beta^0_{N_0,-1}\|_{2,1} \tag{5.72}$$

$$= \|\hat{\Delta}_{N_0,-1}\|_{2,1} \tag{5.73}$$

Thus,

$$\frac{1}{2N}\|X\hat{\Delta}\|_2^2 \leq (\nu_1+\nu_2-\lambda_1)\|\hat{\Delta}_{S_0^c}\|_{2,1} + (\nu_1+\nu_2)\|\hat{\Delta}_{S_0}\|_{2,1} \tag{5.74}$$

$$+ (\nu_2-\lambda_2)\|\hat{\Delta}_{N_0^c,-1}\|_{2,1} + \nu_2\|\hat{\Delta}_{N_0,-1}\|_{2,1} \tag{5.75}$$

$$+ \lambda_1\|\hat{\Delta}_{S_0}\|_{2,1} + \lambda_2\|\hat{\Delta}_{N_0,-1}\|_{2,1} \tag{5.76}$$

$$= (\nu_1+\nu_2+\lambda_1)\|\hat{\Delta}_{S_0}\|_{2,1} + (\nu_2+\lambda_2)\|\hat{\Delta}_{N_0,-1}\|_{2,1} \tag{5.77}$$

$$+ (\nu_1+\nu_2-\lambda_1)\|\hat{\Delta}_{S_0^c}\|_{2,1} + (\nu_2-\lambda_2)\|\hat{\Delta}_{N_0^c,-1}\|_{2,1} \tag{5.78}$$

Thus,

$$\frac{1}{2N}\|X\hat{\Delta}\|_2^2 + (\lambda_1-\nu_1-\nu_2)\|\hat{\Delta}_{S_0^c}\|_{2,1} + (\lambda_2-\nu_2)\|\hat{\Delta}_{N_0^c,-1}\|_{2,1} \tag{5.79}$$

$$\leq (\nu_1+\nu_2+\lambda_1)\|\hat{\Delta}_{S_0}\|_{2,1} + (\nu_2+\lambda_2)\|\hat{\Delta}_{N_0,-1}\|_{2,1} \tag{5.80}$$

Take $\lambda_1 = 2(\nu_1 + \nu_2)$ and $\lambda_2 = 2\nu_2$, then the above inequality implies

$$\frac{1}{2N}\|X\hat{\Delta}\|_2^2 + \frac{1}{2}\lambda_1\|\hat{\Delta}_{S_0^c}\|_{2,1} + \frac{1}{2}\lambda_2\|\hat{\Delta}_{N_0^c,-1}\|_{2,1} \leq \frac{3}{2}\lambda_1\|\hat{\Delta}_{S_0}\|_{2,1} + \frac{3}{2}\lambda_2\|\hat{\Delta}_{N_0,-1}\|_{2,1} \tag{5.81}$$

Notice that (5.81) implies that $\hat{\Delta}$ satisfies (5.56).

Adding $\frac{1}{2}\lambda_1\|\hat{\Delta}_{S_0}\|_{2,1} + \frac{1}{2}\lambda_2\|\hat{\Delta}_{N_0,-1}\|_{2,1}$ to both sides of (5.81), we get

$$\frac{1}{2N}\|X\hat{\Delta}\|_2^2 + \frac{1}{2}\lambda\Omega(\hat{\Delta}) \leq 2\lambda_1\|\hat{\Delta}_{S_0}\|_{2,1} + 2\lambda_2\|\hat{\Delta}_{N_0,-1}\|_{2,1} \tag{5.82}$$

Now we can bound $\frac{1}{N}\|X\hat{\Delta}\|_2^2 + \lambda\Omega(\hat{\Delta})$ if the compatibility condition holds. With this compatibility condition and using the fact that $\forall u, v,\ 4uv \leq u^2 + 4v^2$, we have

$$\begin{aligned}
\frac{1}{N}\|X\hat{\Delta}\|_2^2 + \lambda\Omega(\hat{\Delta}) \leq & 4\lambda_1\|\hat{\Delta}_{S_0}\|_{2,1} + 4\lambda_2\|\hat{\Delta}_{N_0,-1}\|_{2,1} && (5.83)\\
\leq & 4\lambda_1\sqrt{\frac{s_0}{N\phi_{S_0}^2}}\|X\hat{\Delta}\|_2 + 4\lambda_2\sqrt{\frac{n_0}{N\phi_{N_0}^2}}\|X\hat{\Delta}\|_2 && (5.84)\\
= & 4\left(2\lambda_1\sqrt{\frac{s_0}{\phi_{S_0}^2}}\right)\frac{\|X\hat{\Delta}\|_2}{2\sqrt{N}} + 4\left(2\lambda_2\sqrt{\frac{n_0}{N\phi_{N_0}^2}}\right)\frac{\|X\hat{\Delta}\|_2}{2\sqrt{N}} && (5.85)\\
\leq & \frac{1}{4N}\|X\hat{\Delta}\|_2^2 + 16\frac{\lambda_1^2 s_0}{\phi_{S_0}^2} + \frac{1}{4N}\|X\hat{\Delta}\|_2^2 + 16\frac{\lambda_2^2 n_0}{\phi_{N_0}^2} && (5.86)
\end{aligned}$$

Thus,

$$\begin{aligned}
\frac{1}{2N}\|X\hat{\Delta}\|_2^2 + \lambda\Omega(\hat{\Delta}) \leq & 16\left(\frac{\lambda_1^2 s_0}{\phi_{S_0}^2} + \frac{\lambda_2^2 n_0}{\phi_{N_0}^2}\right) && (5.87)\\
\frac{1}{N}\|X\hat{\Delta}\|_2^2 + 2\lambda\Omega(\hat{\Delta}) \leq & 32\left(\frac{\lambda_1^2 s_0}{\phi_{S_0}^2} + \frac{\lambda_2^2 n_0}{\phi_{N_0}^2}\right) && (5.88)
\end{aligned}$$

This means,

$$\begin{aligned}
\frac{1}{N}\|X\hat{\Delta}\|_2^2 \leq & 32\lambda^2\left(\frac{\alpha^2 s_0}{\phi_{S_0}^2} + \frac{(1-\alpha)^2 n_0}{\phi_{N_0}^2}\right) && (5.89)\\
\Omega(\hat{\Delta}) \leq & 16\lambda\left(\frac{\alpha^2 s_0}{\phi_{S_0}^2} + \frac{(1-\alpha)^2 n_0}{\phi_{N_0}^2}\right) && (5.90)
\end{aligned}$$

Table 5.1: RMSE for synthetic dataset in Example 4. Mean RMSE $\pm$ one standard deviation is shown.

Model	$\sigma^2 = 1$	$\sigma^2 = 2$	$\sigma^2 = 4$	$\sigma^2 = 8$	$\sigma^2 = 16$
Lasso Refit	2.9988±0.0596	3.1747±0.0683	3.4589±0.0685	4.0175±0.0469	4.9031±0.0528
SPLAM Refit	**1.0153±0.0180**	**1.4406±0.0293**	**2.0142±0.0315**	**2.8395±0.0646**	**4.0139±0.0432**
SpAM Refit	1.0189±0.0194	1.4442±0.0308	2.0221±0.0313	2.8463±0.0652	4.0296±0.0501
Lasso	3.0019±0.0610	3.1759±0.0666	3.4618±0.0682	4.0186±0.0453	4.9060±0.0507
SPLAM	1.1079±0.0942	1.5269±0.0433	2.1673±0.1760	2.9054±0.0628	4.0675±0.0479
SpAM	1.1127±0.0568	1.6386±0.0201	2.2135±0.1693	3.0197±0.1935	4.3546±0.1617

By choosing $x = \log p$, we can take $\lambda \geq 2(\tilde{\nu}_1^2 + 2\tilde{\nu}_2^2) = 2(1 + 2\sqrt{6})\sigma\sqrt{\log p/N}$ and $\alpha = (\tilde{\nu}_1^2 + \tilde{\nu}_2^2)/(\tilde{\nu}_1^2 + 2\tilde{\nu}_2^2) = (1 + \sqrt{6})/(1 + 2\sqrt{6})$, with probability at least $1 - 4/p$, we have

$$\frac{1}{N}\|X\hat{\beta} - X\beta^0\|_2^2 \leq 32\lambda^2 \left(\frac{\alpha^2 s_0}{\phi_{S_0}^2} + \frac{(1-\alpha)^2 n_0}{\phi_{N_0}^2}\right) \tag{5.91}$$

$$\Omega(\hat{\beta} - \beta^0) \leq 16\lambda \left(\frac{\alpha^2 s_0}{\phi_{S_0}^2} + \frac{(1-\alpha)^2 n_0}{\phi_{N_0}^2}\right) \tag{5.92}$$

□

5.5 Scalability

When the design matrix is sparse, BCGD and BCD for fitting GLMs are extremely fast since we only need to scan the list of non-zero entries. However, this nice property is unfortunately not necessarily preserved given our representation of the features; for $i > 1$, $b_i(0)$ is not necessarily zero anymore. For example, in cubic spline described in Section 5.2.1, $(0 - x_{j\cdot}^*)_+^3 \neq 0$ whenever the knot $x_{j\cdot}^* < 0$.

Nevertheless, notice that the design matrix is now a *data-sparse* matrix, where we have a lot of $b_i(0)$s instead of 0s. Therefore, we propose a general technique for solving convex problems on data-sparse design matrix. Assume the dimension is p and the convex regularization is only applied on weight vector $w \in \mathbb{R}^p$.

Let w_0 be the intercept and consider the following convex optimization problem.

OP 1.

$$\min_{w,w_0} F(y, \sum_j x_j w_j + w_0), \tag{5.93}$$

Let z_j be the dominating elements for variable x_j, The above optimization problem can be rewritten as,

$$\min_{w,w_0} F(y, \sum_j (x_j - z_j) w_j + \sum_j z_j w_j + w_0) \tag{5.94}$$

Let $w_0' = \sum_j z_j w_j + w_0$ and $x_j' = x_j - z_j$. The equivalent optimization problem becomes,

OP 2.

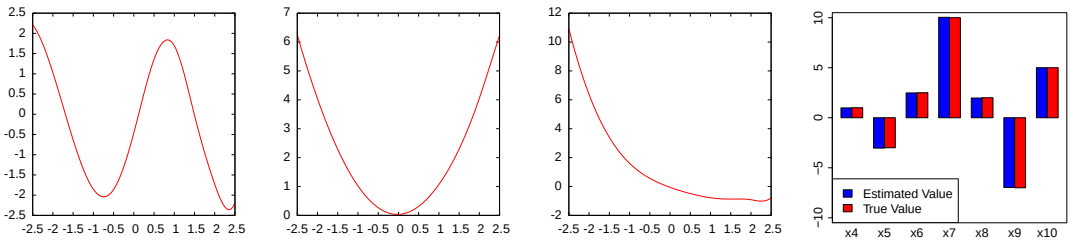
$$\min_{w,w_0'} F(y, \sum_j x_j' w_j + w_0'), \tag{5.95}$$

It is easy to see OP 2 now enjoys the sparsity of the design matrix and therefore the fitting procedure only needs to scan the list of non-zero entries.

For regression problems using BCD as described in Section 5.3.1, although the design matrix can be made sparse using this method, Q_j after QR decomposition is unfortunately not sparse. When the dimensionality is large, storing all (dense) Q_js would be intractable. Fortunately, all we need is $Q_j^T r_j$ and observing that $Q_j^T = (R_j^{-1})^T X_j^T$, we can instead compute $(R_j^{-1})^T (X_j^T r_j)$ during the iteration. This only involves a sparse matrix multiplication followed by a constant time matrix multiplication since $(R_j^{-1})^T$ is M-by-M, which does not increase the complexity of each BCD iteration.

5.6 Experiments

In this section, we report experimental results for SPLAM. For all our experiments, we use 10 knots for cubic splines and choose the best parameters on the



(a) f_1 (b) f_2 (c) f_3 (d) Weights

Figure 5.2: Estimated component functions and weights for synthetic dataset in Example 4.

held-out validation set and report model performance on test set.

5.6.1 Synthetic Problem

To understand the properties of SPLAM, we generate 10,000 points using the synthetic function in Example 4. In this experiment, we create an additional 90 random features and vary the noise variance $\sigma^2 = 1, 2, 4, 8, 16$. The first 3 nonlinear features are generated uniformly in $[-2.5, 2.5]$, and all other features are uniformly in $[0, 1]$.

We compare SPLAM with Lasso [62] and SpAM [58]. Lasso builds a linear model with ℓ_1 regularization and SpAM constructs a sparse additive model. For SPLAM, we consider $\alpha = 0.05k$, $k = 1, ..., 20$. Notice that SPLAM becomes SpAM when $\alpha = 1$. For all these three methods, we consider full regularization paths with 100 λs in log scale. In our experiments, this range is sufficient to find the optimal model structure. Best model parameters are chosen using the validation set. Thus, we have the full spectrum of sparse models from a pure linear model to a pure additive model.

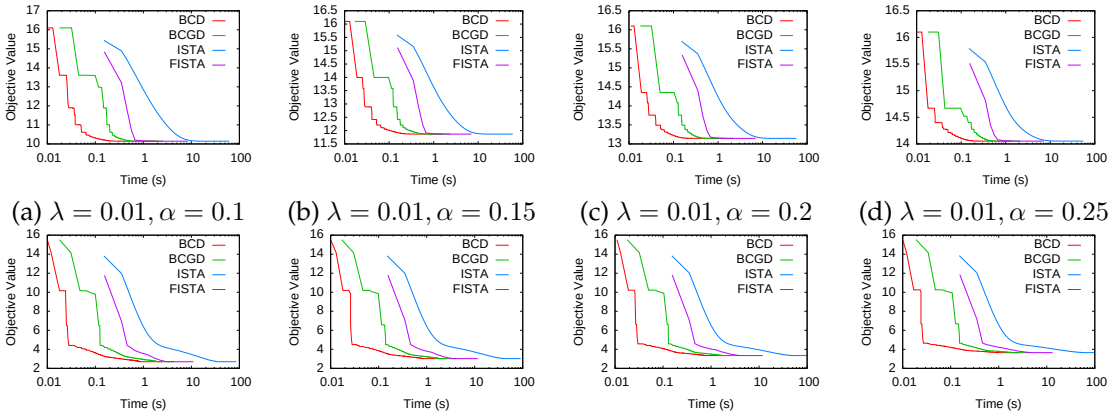
Accuracy

Table 5.1 summarizes the experimental results. For each model, we report root mean squared error (RMSE) with refitting strategy on and off. We begin by observing that the RMSE is always better when refitting strategy is used. In this example, we have sufficient data for reliable estimation so the strong priors that regularization parameters impose can lead to a suboptimal model. Thus, we by default use the refitting strategy when fitting those models unless otherwise stated. Second, since we have 3 nonlinear components in the ground truth model, as expected, both SPLAM and SpAM outperform Lasso. However, as we carefully control the complexity in SPLAM, SPLAM is always better than SpAM. Actually, when refitting is used, SPLAM outperforms SpAM on *every* cross validation set. As the noise level goes up, performance of all models degrades as one would expect. This shows the excellent accuracy of SPLAM in practice when the true model structure is not known a priori.

Estimated Components

We plot estimated components in Figure 5.2 for $\sigma^2 = 1$. Figure 5.2 (a), (b), and (c) visualize the nonlinear components for f_1, f_2, and f_3, respectively. We can see the estimated shape of the component function is very plausible. Figure 5.2 (d) shows the estimated weights on x_4 to x_{10} together with the true weights. Similarly, the estimated weights are very close to the ground truth.

We also perform 10 trials by generating random datasets of same size and evaluate the support by three models. Both SPLAM and SpAM can always find the correct support while Lasso sometimes makes mistakes with precision 0.96 ± 0.05 and recall 0.98 ± 0.04, where precision is ratio of the number of true selected variables over the number of selected variables, and recall is the ratio



(a) $\lambda = 0.01, \alpha = 0.1$ (b) $\lambda = 0.01, \alpha = 0.15$ (c) $\lambda = 0.01, \alpha = 0.2$ (d) $\lambda = 0.01, \alpha = 0.25$

(e) $\lambda = 0.001, \alpha = 0.1$ (f) $\lambda = 0.001, \alpha = 0.15$ (g) $\lambda = 0.001, \alpha = 0.2$ (h) $\lambda = 0.001, \alpha = 0.25$

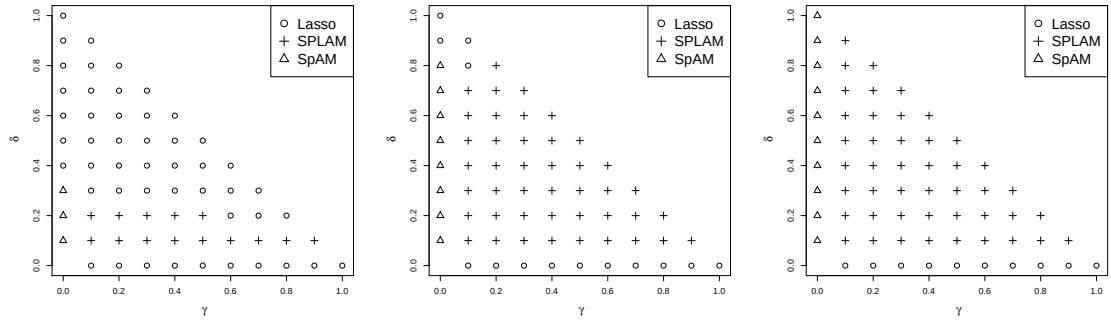
Figure 5.3: Objective value vs. running time for synthetic dataset in Example 4.

of the number of true selected variables over the number of true variables.

Running Time

In this section, we compare our BCGD algorithm and BCD algorithm with ISTA and FISTA [11]. We report running time of all the methods on a single core. For BCGD, ISTA, and FISTA, we start with a same initial step size. For fair comparison, we turn off the active set strategy in BCGD and BCD, and we directly use the design matrix after QR decomposition so that all methods are applied to the same optimization problem.

Figure 5.3 illustrates the running time for all methods using the same synthetic dataset in Example 4 for different combinations of λ and α. As expected, FISTA converges much faster than ISTA. However, the BCGD algorithm is faster than both of the these methods. This is because BCGD uses more information than the first-order methods. In addition, we can see that BCD further speeds up the optimization since there is no step size in BCD; this not only solves exactly the subproblem but also avoids the possibility of dampening the step size and repeating the computation on the same block.



$N_{train} = 200$ $N_{train} = 500$ $N_{train} = 1000$

Figure 5.4: Results for simulation in Section 5.6.2. Each plot shows the wining model for a given (δ, γ) pair.

5.6.2 Simulation

In this section, we perform a large-scale simulation to gain deeper insights into the Lasso, SPLAM and SpAM. We consider the models with $p = 100$ features: $y = \sum_{i\in\mathcal{I}} x_i + \sum_{j\in\mathcal{J}} \sin(x_j) + \epsilon$, where $\epsilon \sim \mathcal{N}(0, 1)$, $\mathcal{I} \cap \mathcal{J} = \emptyset$. We use two parameters γ and δ to control the cardinality of $\mathcal{I}$ and $\mathcal{J}$, respectively, i.e., $|\mathcal{I}| = \gamma p$ and $|\mathcal{J}| = \delta p$. We choose $\gamma = 0.0, 0.1, ..., 1.0$ and $\delta = 0.0, 0.1, ..., 1.0$ ($\gamma + \delta \leq 1$) and for each (γ, δ) pair, we generate 10 different models. For each of those models, we generate N_{train} points for training, N_{valid} points for validation and N_{test} points for testing. We consider three different settings of simulations, $(N_{train}, N_{valid}, N_{test}) = (200, 100, 100), (500, 100, 100), (1000, 200, 200)$. Best model parameters are chosen using the validation set and model accuracy is evaluated as the average RMSE of 10 models on test sets.

Figure 5.4 shows the results for the simulations. For each (γ, δ) pair, we plot which model wins on average. It is clear that for pure linear ($\delta = 0$) and pure additive ($\gamma = 0$), SPLAM has no advantage over Lasso or SpAM.

When $N_{train} = 200$, both SpAM and SPLAM overfit significantly when there are a lot of nonlinear components, since a large number of nonlinear compo-

Table 5.2: Datasets.

Dataset	Size	Test	Dimension	%Pos
Spambase	4601	920	58	39.40
Gisette	6000	1200	5001	50.00
RCV1	697641	418584	47236	52.46
Pantheon	62849	37709	10000	50.00

nents leads to a large parameter space and this small amount of data is not enough for reliable estimates. Lasso wins over the other methods on most of the cases by trading off bias for accuracy. SPLAM outperforms Lasso in regimes with a mixture of small nonlinear components and a reasonable number of linear components. When we increase the number of data points in the training set ($N_{train} = 500$), more reliable estimates can be obtained so SpAM wins back from Lasso on cases where we only have nonlinear components, since Lasso as a linear model is incapable of estimating nonlinear effects present in the data. Again, we see Lasso is still the best when there are a lot of nonlinear components since we are back in the regime where the data cannot support the large number of parameters for reliable estimation. SPLAM, however, becomes the best on most settings since it can better model the mixture of linear and nonlinear effects when there is enough data. Not surprisingly, when there are enough data ($N_{train} = 1000$), SPLAM dominates all cases when there are both linear and nonlinear components. This is because Lasso is unable to model nonlinear effects and because SPLAM carefully controls the model complexity so that the variance of SPLAM is lower than SpAM with slightly more bias.

5.6.3 Real Problems

In this section, we report experimental results on several real classification problems. We choose datasets with different dimensions and sizes. Characteristics of

Table 5.3: Classification error (%) for datasets in Section 5.6.3. Each cell contains the mean error $\pm$ one standard deviation.

Model	Spambase	Gisette	RCV1	Pantheon
ℓ_1-LR	7.38$\pm$0.87	2.52$\pm$0.71	2.66$\pm$0.02	9.07$\pm$0.10
SPLAM	**6.35$\pm$0.82**	**2.32$\pm$0.62**	**2.55$\pm$0.01**	**9.00$\pm$0.11**
SpAM	6.59$\pm$0.96	2.85$\pm$0.48	2.70$\pm$0.02	9.25$\pm$0.12

the datasets are summarized in Table 5.2. Table 5.3 presents the predictive performance for ℓ_1-regularized logistic regression, SPLAM, and SpAM on 5-fold cross validations.

Email Classification

We first consider a classification problem for detecting spam emails (Spambase) [33]. The features measure the percentage of specific words or characters in the email, the average and maximum lengths of uninterrupted upper case letters, and the total number of such letters.

We see from Table 5.3 that by allowing features to act nonlinearly, the error of SpAM decreases substantially compared to ℓ_1-regularized logistic regression. However, by explicitly setting some of the variables to stay linear, SPLAM further outperforms SpAM.

One property of this dataset is that it is very sparse. On average 77.41% entries are 0. We use this dataset to demonstrate our technique for training on data-sparse problems. Figure 5.5 shows the running time for dense and sparse optimization with different λ and α combinations. Clearly, by exploiting the data sparsity, the running time is significantly reduced.

Handwritten Digit Recognition

We use the "Gisette" dataset constructed from the MNIST data. The problem is to separate the highly confusible digits "4" and "9". We adopt the version

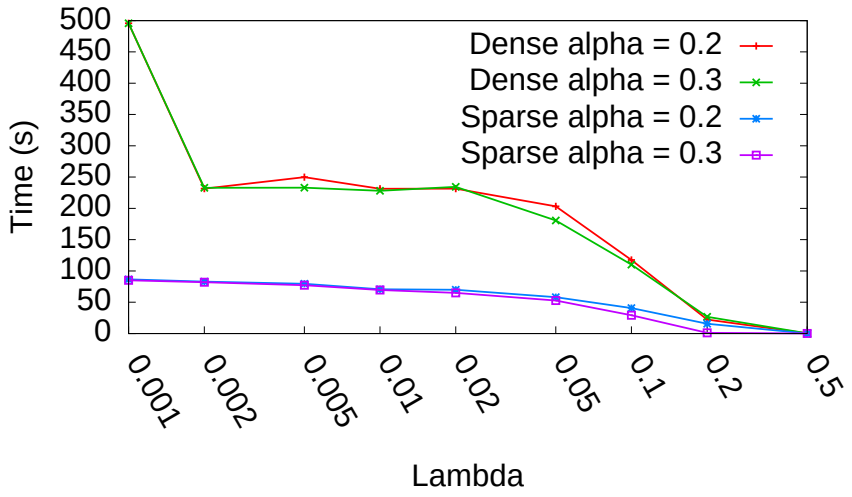


Figure 5.5: Running time for Spambase.

that was used in the NIPS 2003 feature selection challenge [7]. For the purpose of the feature selection challenge, pixels were samples at random in the middle top part of the feature containing the information necessary to disambiguate "4" from "9" and higher order features were created as products of these pixels to force the problem in a higher dimensional feature space. Distractor features with no predictive power were also added.

Since the dimension of this dataset is significantly larger than the previous dataset while the size of the dataset remains similar, we expect SpAM to overfit as shown in Table 5.3. In this and the following experiments, we turn off refitting strategy as this results in a better predictive performance for all models. In our experiments, the best SpAM model that we can get is always worse than ℓ_1-regularized logistic regression on each cross validation set while our SPLAM outperforms ℓ_1-regularized logistic regression on most cross validation sets.

Our SPLAM selects about 400 features, with about 10 of them being nonlinear and the rest being linear. This confirms that by allowing a small number

of features to act nonlinearly, we can further improve the classification performance, and yet by setting most of features as linear, we effectively control the complexity and avoid overfitting.

Text Categorization

Text categorization is an important task for many natural language processing applications. We use Reuters Corpus Volume I (RCV1) which involves binary classification [41]. All features are real valued, which allows us to model nonlinear effects. It is important to note that for a number of high dimensional datasets, features are binarized (in order to save storage space and to reduce memory footprint). On those datasets with only binary features, SPLAM has no advantage over linear models since we cannot model nonlinear effects from binary features.

Table 5.3 shows the predictive performance of the three models. We see that SPLAM outperforms the others. This suggests that in high dimensions, although ℓ_1-regularized linear model is popular, there is extra accuracy that can be obtained if some features are allowed to be nonlinearly transformed. However, if all features are allowed to be nonlinearly transformed, such as in SpAM, the model will overfit and a suboptimal model is obtained.

Image Matching

Many new computer vision applications are utilizing large-scale datasets of places derived from the many billions of photos on the Web. Image matching is a central procedure to those applications which tests whether two images are geometrically consistent [48]. Since image matching is an expensive procedure, image pairs are usually pre-filtered with a lightweight classification procedure

to estimate whether two images are likely to pass the geometric verification. In this study, we use the "Pantheon" dataset in [48]. Each image is represented using bag-of-visual-words model with a vocabulary of 10,000 visual words.

From Table 5.3 we again observe that by carefully controlling the complexity of the model, SPLAM has better predictive performance than the other two models. On average ℓ_1-regularized logistic regression selects 1538 features while SPLAM selects 1529 features with 10 of them being nonlinear. This also implies that when making predictions using SPLAM, there is virtually no extra cost of computing the predictions compared to linear models. For high dimensional datasets, it is usually hard to model nonlinearities or higher order feature interactions [47] because these complex models can be easily overfit. Therefore, high bias models such as ℓ_1-regularized linear models are usually preferred. Our SPLAM further improves the accuracy from linear models by modeling nonlinear effects for a small subset of intelligently chosen features.

5.7 Conclusion

In this chapter, we introduce generalized sparse partially linear additive models (SPLAMs), where in addition to variable selection, each variable can stay linear as in standard GLMs, or it can be allowed to act nonlinearly as in standard GAMs when there is enough evidence in the data. Thus, SPLAM offers lower complexity than GAM but has more flexibility than GLM, and provides a data-driven approach to automatically balance the bias and the variance.

We model the problem as a convex hierarchical sparse regularization problem and propose two optimization problems using block coordinate gradient descent and block coordinate descent, respectively. We provide the global convergence analysis for the block coordinate gradient descent algorithm and a sta-

tistical analysis of the theoretical properties of SPLAM. Finally we propose a general technique for optimization on data-sparse problems. Our experiments demonstrate that SPLAM can effectively and accurately find relevant components with proper complexity and is very competitive for additive modeling. In particular, on high dimensional datasets SPLAM improves accuracy from the popular linear model by allowing a small set of features to act nonlinearly. This significantly improves model intelligibility.

CHAPTER 6

CONCLUSION

Different supervised learning models have different bias-variance tradeoffs. For low dimensional problems, low-bias models such boosted trees or SVMs with RBF kernels are very accurate but are unfortunately no longer interpretable by the users. For high dimensional problems, high-bias models such as regularized linear/logistic regressions are usually preferred over other models because of the curse of dimensionality and the exponentially growing hypothesis space but it is not clear whether we could further improve the accuracy from those high-bias models.

This dissertation revisits the classic additive modeling and is an attempt to better balance the bias and variance to build interpretable models. We present a family of additive models called intelligible models, which effectively recover the low dimensional additive structures. Those low dimensional additive components provide the opportunities for data scientists to investigate each simple component individually, and therefore the interpretability is significantly improved. The key contribution is a series of algorithms that build intelligible models with large-scale empirical evaluations. We first present a large-scale empirical study of various methods for fitting GAMs. We demonstrate empirically that gradient boosting with shallow bagged trees yield the best accuracy. In addition, we propose a very efficient method of detecting pairwise feature interactions that scales to thousands of features. With a large-scale empirical study, we show that models with low dimensional additive components (one- and two-dimensional components) are as accurate as complex models such as random forests. Finally, we develop a method to carefully control the complexity of the intelligible models by feature selection and intelligently deciding

whether the selected term is linear or nonlinear, and show that on high dimensional problems we can further improve the accuracy from the popular linear models by allowing a small set of features to act nonlinearly.

BIBLIOGRAPHY

[1] http://archive.ics.uci.edu/ml/.

[2] http://www.liaad.up.pt/~ltorgo/Regression/DataSets.html.

[3] http://www.cs.toronto.edu/~delve/data/datasets.html.

[4] http://osmot.cs.cornell.edu/kddcup/.

[5] http://additivegroves.net.

[6] http://research.microsoft.com/en-us/projects/mslr/.

[7] http://www.nipsfsc.ecs.soton.ac.uk/.

[8] http://www-stat.stanford.edu/~jhf/R-RuleFit.html.

[9] F.R. Bach. Consistency of the group lasso and multiple kernel learning. *The Journal of Machine Learning Research*, 9:1179–1225, 2008.

[10] E. Bauer and R. Kohavi. An empirical comparison of voting classification algorithms: Bagging, boosting, and variants. *Machine learning*, 36(1):105–139, 1999.

[11] A. Beck and M. Teboulle. A fast iterative shrinkage-thresholding algorithm for linear inverse problems. *SIAM Journal on Imaging Sciences*, 2(1):183–202, 2009.

[12] Yoshua Bengio. Learning deep architectures for ai. *Foundations and trends® in Machine Learning*, 2(1):1–127, 2009.

[13] H. Binder and G. Tutz. A comparison of methods for the fitting of generalized additive models. *Statistics and Computing*, 18(1):87–99, 2008.

[14] C.M Bishop. *Pattern recognition and machine learning*. Springer New York, 2007.

[15] L. Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[16] L. Breiman and J.H. Friedman. Estimating optimal transformations for multiple regression and correlation. *Journal of the American Statistical Association*, pages 580–598, 1985.

[17] L. Breiman, J.H. Friedman, R.A. Olshen, and C.J. Stone. Classification and regression trees. *Wadsworth International Group*, 1984.

[18] P. Bühlmann and S. Van De Geer. *Statistics for high-dimensional data: methods, theory and applications*. Springer, 2011.

[19] R. Caruana, N. Karampatziakis, and A. Yessenalina. An empirical evaluation of supervised learning in high dimensions. In *ICML*, 2008.

[20] R. Caruana and A. Niculescu-Mizil. An empirical comparison of supervised learning algorithms. In *ICML*, 2006.

[21] P.K. Chan and S.J. Stolfo. Toward scalable learning with non-uniform class and cost distributions: A case study in credit card fraud detection. In *KDD*, 1998.

[22] R. Chen, H. Liang, and J. Wang. Determination of linear components in additive models. *Journal of Nonparametric Statistics*, 23(2):367–383, 2011.

[23] C. Cortes and V. Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

[24] G. Forman, M. Scholz, and S. Rajaram. Feature shaping for linear svm classifiers. In *KDD*, 2009.

[25] J. Friedman, T. Hastie, and R. Tibshirani. Additive logistic regression: a statistical view of boosting (with discussion and a rejoinder by the authors). *Annals of Statistics*, 28:337–407, 2000.

[26] J.H. Friedman. Greedy function approximation: a gradient boosting machine. *Annals of Statistics*, 29:1189–1232, 2001.

[27] J.H. Friedman. Stochastic gradient boosting. *Computational Statistics and Data Analysis*, 38:367–378, 2002.

[28] J.H. Friedman, T. Hastie, and R. Tibshirani. Regularization paths for generalized linear models via coordinate descent. *Journal of Statistical Software*, 33(1):1, 2010.

[29] J.H. Friedman and B. Popescu. Predictive learning via rule ensembles. *The Annals of Applied Statistics*, pages 916–954, 2008.

[30] I. Guyon and A. Elisseeff. An introduction to variable and feature selection. *The Journal of Machine Learning Research*, 3:1157–1182, 2003.

[31] W. Härdle and H. Liang. *Partially linear models*. Springer, 2007.

[32] T. Hastie and R. Tibshirani. *Generalized additive models*. Chapman & Hall/CRC, 1990.

[33] T Hastie, R. Tibshirani, and J.H. Friedman. *The elements of statistical learning (2nd edition)*. Springer New York, 2009.

[34] G. Hooker. Discovering additive structure in black box functions. In *KDD*, 2004.

[35] G. Hooker. Generalized functional anova diagnostics for high-dimensional functions of dependent variables. *Journal of Computational and Graphical Statistics*, 16(3):709–732, 2007.

[36] R. Jenatton, J. Mairal, F.R. Bach, and G.R. Obozinski. Proximal methods for sparse hierarchical dictionary learning. In *ICML*, 2010.

[37] T. Joachims. Training linear svms in linear time. In *KDD*, 2006.

[38] R. Kelley Pace and R. Barry. Sparse spatial autoregressions. *Statistics & Probability Letters*, 33(3):291–297, 1997.

[39] S. Kim and E.P. Xing. Tree-guided group lasso for multi-task regression with structured sparsity. In *ICML*, 2009.

[40] B. Krishnapuram, L. Carin, M.A. Figueiredo, and A.J. Hartemink. Sparse multinomial logistic regression: Fast algorithms and generalization bounds. *Transactions on Pattern Analysis and Machine Intelligence*, 27(6):957–968, 2005.

[41] D.D. Lewis, Y. Yang, T.G. Rose, and F. Li. Rcv1: A new benchmark collection for text categorization research. *The Journal of Machine Learning Research*, 5:361–397, 2004.

[42] P. Li. Abc-boost: Adaptive base class boost for multi-class classification. In *ICML*, 2009.

[43] P. Li, C. Burges, and Q. Wu. Mcrank: Learning to rank using multiple classification and gradient boosting. In *NIPS*, 2007.

[44] J. Liu and J. Ye. Moreau-yosida regularization for grouped tree structure learning. In *NIPS*, 2010.

[45] W.Y. Loh. Regression trees with unbiased variable selection and interaction detection. *Statistica Sinica*, 12(2):361–386, 2002.

[46] Y. Lou, R. Caruana, and J. Gehrke. Intelligible models for classification and regression. In *KDD*, 2012.

[47] Y. Lou, R. Caruana, J. Gehrke, and G. Hooker. Accurate intelligible models with pairwise interactions. In *KDD*, 2013.

[48] Y. Lou, N. Snavely, and J. Gehrke. Matchminer: Efficient spanning structure mining in large image collections. In *ECCV*, 2012.

[49] C. D Manning, P. Raghavan, and H. Schütze. *Introduction to information retrieval*. Cambridge University Press Cambridge, 2008.

[50] L. Meier, S. Van De Geer, and P. Bühlmann. The group lasso for logistic regression. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 70(1):53–71, 2008.

[51] T.M. Mitchell. *Machine learning*. McGraw Hill, 1997.

[52] Y. Nesterov. Gradient methods for minimizing composite objective function. *CORE report*, 2007.

[53] A.Y Ng and M.I Jordan. On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes. In *NIPS*, 2002.

[54] B. Panda, J.S. Herbach, S. Basu, and R.J. Bayardo. Planet: massively parallel learning of tree ensembles with mapreduce. *PVLDB*, 2009.

[55] Z. Qin, K. Scheinberg, and D. Goldfarb. Efficient block-coordinate descent algorithms for the group lasso. *Mathematical Programming Computation*, pages 1–27, 2010.

[56] J.R. Quinlan. Combining instance-based and model-based learning. In *ICML*, 1993.

[57] G Raskutti, M.J. Wainwright, and B. Yu. Restricted eigenvalue properties for correlated gaussian designs. *The Journal of Machine Learning Research*, 11:2241–2259, 2010.

[58] P. Ravikumar, H. Liu, J. Lafferty, and L. Wasserman. Sparse additive models. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 71(5):1009–1030, 2009.

[59] D. Sorokina, R. Caruana, and M. Riedewald. Additive groves of regression trees. In *ECML*, 2007.

[60] D. Sorokina, R. Caruana, M. Riedewald, and D. Fink. Detecting statistical interactions with additive groves of trees. In *ICML*, 2008.

[61] R Szeliski. *Computer vision: algorithms and applications*. Springer, 2010.

[62] R. Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 267–288, 1996.

[63] P. Tseng and S. Yun. A coordinate gradient descent method for nonsmooth separable minimization. *Mathematical Programming*, 117(1-2):387–423, 2009.

[64] S. Tyree, K.Q. Weinberger, K. Agrawal, and J. Paykin. Parallel boosted regression trees for web search ranking. In *WWW*, 2011.

[65] S. van de Geer and J. Lederer. The lasso, correlated design, and improved oracle inequalities. In *From Probability to Statistics and Back: High-Dimensional Models and Processes–A Festschrift in Honor of Jon A. Wellner*, pages 303–316. Institute of Mathematical Statistics, 2013.

[66] S.M. Weiss and N. Indurkhya. Rule-based machine learning methods for functional prediction. *Journal of Artificial Intelligence Research*, 3:383–403, 1995.

[67] S.N. Wood. Thin plate regression splines. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 65(1):95–114, 2003.

[68] S.N. Wood. *Generalized additive models: an introduction with R*. CRC Press, 2006.

[69] H. Xie and J. Huang. Scad-penalized regression in high-dimensional partially linear models. *The Annals of Statistics*, pages 673–696, 2009.

[70] P. Zhao, G. Rocha, and B. Yu. The composite absolute penalties family for grouped and hierarchical variable selection. *The Annals of Statistics*, 37(6A):3468–3497, 2009.